

1. Vision and Image Characteristics useful for Compression
 1. [Introduction](#)
 2. [Human Vision](#)
 3. [Image Characteristics](#)
2. A Basic Image Compression Example
 1. [A Basic Image Compression Example](#)
 2. [The Haar Transform](#)
 3. [Entropy](#)
 4. [The Multi-level Haar Transform](#)
 5. [Use of Laplacian PDFs in Image Compression](#)
 6. [Practical Entropy Coding Techniques](#)
3. The DCT and the JPEG Standard
 1. [The Discrete Cosine Transform \(DCT\)](#)
 2. [Fast Algorithms for the DCT](#)
 3. [The 2-dimensional DCT](#)
 4. [Quantisation of DCT Coefficients](#)
 5. [JPEG Entropy Coding](#)
 6. [Sync and Headers](#)
4. Filter Banks and Wavelets
 1. [The 2-band Filter Bank](#)
 2. [Perfect Reconstruction \(PR\)](#)
 3. [The Binary Filter Tree](#)
 4. [Wavelets](#)
 5. [Good Filters / Wavelets](#)
 6. [The 2-D DWT](#)
 7. [Compression Properties of Wavelets](#)
5. Video Compression and Motion Processing
 1. [Motion-Compensated Predictive Coding](#)
 2. [Motion Estimation](#)
 3. [The MPEG Standard](#)

Introduction

Image Coding (often more correctly known as Image Compression) is the art / science of representing images with the **least information** (no. of bits) consistent with achieving an **acceptable image quality / usefulness**.

In order to do this, we try to take advantage of:

1. Physiological characteristics of human vision;
2. Statistical characteristics of typical images;
3. Efficient binary source coding methods.

We shall consider these in turn.

Human Vision

Colours

The human vision system perceives images in colour using receptors on the retina of the eye which respond to three relatively broad colour bands in the regions of red, green and blue (RGB) in the colour spectrum (red, orange, yellow, green, blue, indigo, violet).

Colours in between these are perceived as different linear combinations of RGB. Hence colour TVs and monitors can form almost any perceivable colour by controlling the relative intensities of R, G and B light sources. Thus most colour images which exist in electronic form are fundamentally represented by 3 intensities (R, G and B) at each picture element (pel) position.

The numerical values used for these intensities are usually chosen such that equal increments in value result in approximately equal apparent increases in brightness. In practise this means that the numerical value is approximately proportional to the log of the true light intensity (energy of the wave) - this is **Weber's Law**. Throughout this course, we shall refer to these numerical values as intensities, since for compression it is most convenient to use a subjectively linear scale.

The YUV Colour Space

The eye is much more sensitive to overall intensity (luminance) changes than to colour changes. Usually most of the information about a scene is contained in its luminance rather than its colour (chrominance).

This is why black-and-white (monochrome) reproduction was acceptable for photography and TV for many years until technology provided colour reproduction at a sufficient cheap price to make its modest advantages worth having.

The luminance (Y) of a pel may be obtained from its RGB components as:

Equation:

$$Y = 0.3R + 0.6G + 0.1B$$

These coefficients are only approximate, and are the values defined in the JPEG Book. In other places values of 0.3, 0.59 and 0.11 are used.

RGB representations of images are normally defined so that if $R = G = B$, the pel is always some shade of gray, and if $Y = R = G = B$ in these cases, the 3 coefficients in [\[link\]](#) should sum to unity.

When Y defines the luminance of a pel, its chrominance is usually defined by U and V such that:

$$U = 0.5(B - Y)$$

Equation:

$$V = 0.625(R - Y)$$

Note that gray pels will always have $U = V = 0$.

The transformation between RGB and YUV colour spaces is linear and may be achieved by a 3×3 matrix C and its inverse:

Equation:

$$\begin{array}{ccc} Y & & R \\ U & = C & G \\ V & & B \end{array}$$

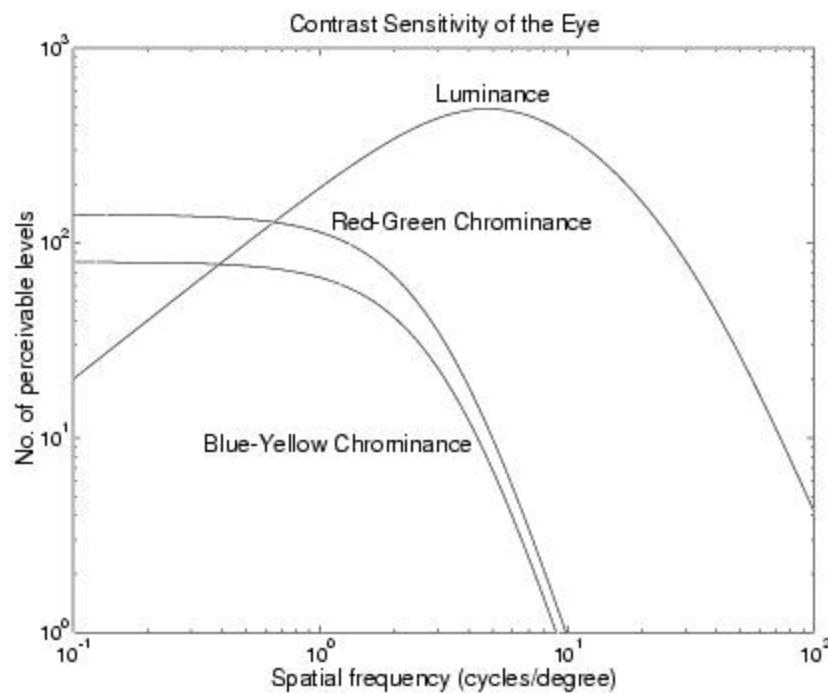
$$\text{where } C = \begin{array}{ccc} 0.3 & 0.6 & 0.1 \\ -0.15 & -0.3 & 0.45 \\ 0.4375 & -0.3750 & -0.0625 \end{array} \text{ and}$$

Equation:

$$\begin{array}{ccc}
 R & & Y \\
 G & = C^{-1} & U \\
 B & & V
 \end{array}$$

$$\begin{array}{ccc}
 1 & 0 & 1.6 \\
 \text{where } C^{-1} = & 1 & -0.3333 & -0.8 \\
 & 1 & 2 & 0
 \end{array}$$

Visual Sensitivity



Sensitivity of the eye to luminance and chrominance intensity changes.

[\[link\]](#) shows the sensitivity of the eye to luminance (Y) and chrominance (U, V) components of images. The horizontal scale is spatial frequency, and

represents the frequency of an alternating pattern of parallel stripes with sinusoidally varying intensity. The vertical scale is the contrast sensitivity of human vision, which is the ratio of the maximum visible range of intensities to the minimum discernible peak-to-peak intensity variation at the specified frequency.

In [\[link\]](#) we see that:

- the maximum sensitivity to Y occurs for spatial frequencies around 5 cycles / degree, which corresponds to striped patterns with a half-period (stripe width) of 1.8 mm at a distance of 1 m (~arm's length).
- The eye has very little response above 100 cycles / degree, which corresponds to a stripe width of 0.1 mm at 1 m. On a standard PC display of width 250 mm, this would require 2500 pels per line! Hence the current SVGA standard of 1024×768 pels still falls somewhat short of the ideal and is limited by CRT spot size. Modern laptop displays have a pel size of about 0.3 mm, but are pleasing to view because the pel edges are so sharp (and there is no flicker).
- The sensitivity to luminance drops off at low spatial frequencies, showing that we are not very good at estimating absolute luminance levels **as long as they do not change with time** - the luminance sensitivity to temporal fluctuations (flicker) does not fall off at low spatial frequencies.
- The maximum chrominance sensitivity is much lower than the maximum luminance sensitivity with blue-yellow (U) sensitivity being about half of red-green (V) sensitivity and about $\frac{1}{6}$ of the maximum luminance sensitivity.
- The chrominance sensitivities fall off above 1 cycle / degree, requiring a much lower spatial bandwidth than luminance.

We can now see why it is better to convert to the YUV domain before attempting image compression. The U and V components may be sampled at a lower rate than Y (due to narrower bandwidth) and may be quantised more coarsely (due to lower contrast sensitivity).

A colour demonstration on the computer will show this effect.

Colour compression Strategy

The 3 RGB samples at each pel are transformed into 3 YUV samples using [\[link\]](#).

Most image compression systems then subsample the U and V information by 2:1 horizontally and vertically so that there is one U and one V pel for each 2×2 block of Y pels. The subsampled U and V pels are obtained by averaging the four U and V samples, from [\[link\]](#). The quarter-size U and V subimages are then compressed using the same techniques as the full-size Y image, except that coarser quantisation may be used for U and V , so the total cost of adding colour may only be about 25% increase in bit rate. Sometimes U and V are subsamples 4:1 each way (16:1 total), giving an even lower cost of colour.

From now on we will mostly be considering compression of the monochrome Y image, and assume that similar techniques will be used for the smaller U and V subimages.

Activity Masking

A final feature of human vision, which is useful for compression, is that the contrast sensitivity to a given pattern is reduced in the presence of other patterns (activity) in the same region. This is known as activity masking.

It is a complicated subject as it depends on the similarity between the given pattern and the background activity. However in general, the higher the variance of the pels in a given region (typically ~ 8 to 16 pels across), the lower is the contrast sensitivity.

Hence compression schemes which adapt the quantisation to local image activity tend to perform better than those which use uniform quantisation.

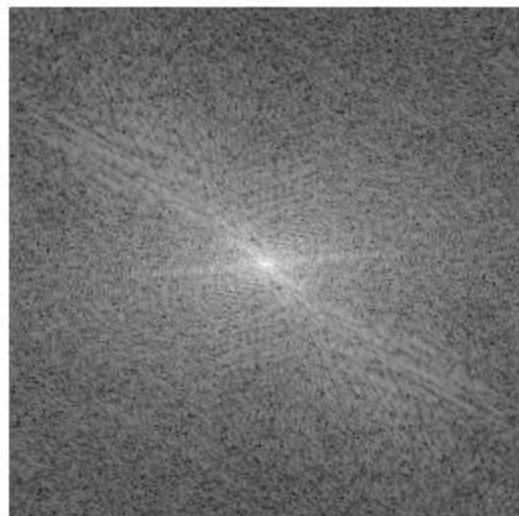
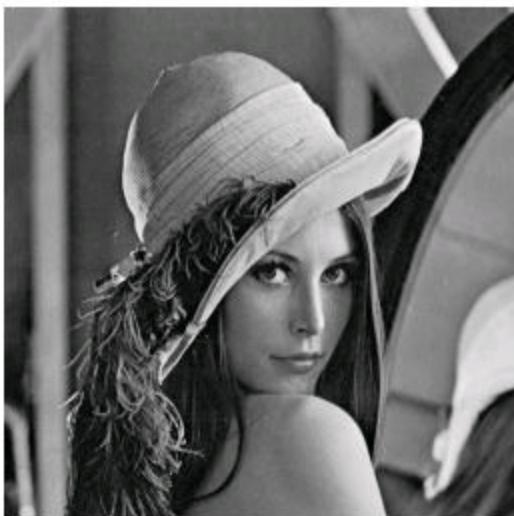
A computer demonstration will show the effect of reduced sensitivity to quantisation effects when noise is added to an image.

Image Characteristics

We now consider statistical characteristics of typical images which can permit compression. If all images comprised dots with uncorrelated random intensities, then each pel would need to be coded independently and we could not achieve any useful gains. However typical images are very different from random dot patterns and significant compression gains are possible.

Some compression can be achieved even if no additional distortion is permitted (**lossless coding**) but much greater compression is possible if some additional distortion is allowed (**lossy coding**). Lossy coding is the main topic of this course but we try to keep the added distortions near or below the human visual sensitivity thresholds discussed previously.

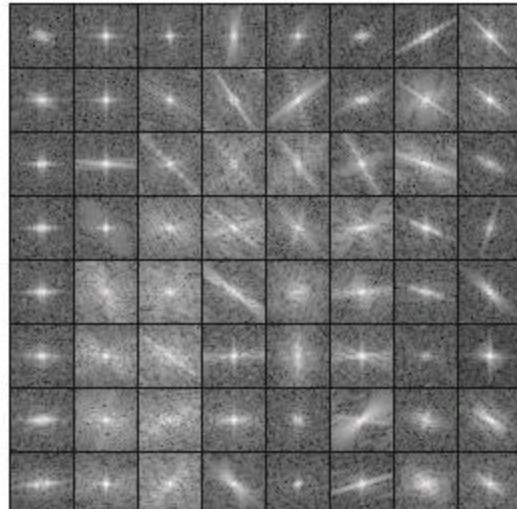
Statistical characteristics of signals can often be most readily appreciated by frequency domain analysis since the power spectrum is the Fourier transform of the autocorrelation function. The 2-D FFT is a convenient tool for analysing images. [\[link\]](#) shows the 256×256 pel 'Lenna' image and its Fourier log power spectrum. Zero frequency is at the centre of the spectrum image and the log scale shows the lower spectral components much more clearly.



256×256 pel 'Lenna' image and its Fourier log power spectrum.

The bright region near the centre of the spectrum shows that the main concentration of image energy is at low frequencies, which implies strong correlation between nearby pels and is typical of real-world images. The diagonal line of spectral energy at about -30° is due to the strong diagonal edges of the hat normal to this direction. Similarly the near-horizontal spectral line comes from the strong near-vertical stripe of hair to the right of the face. Any other features are difficult to distinguish in this global spectrum.

A key property of real-world images is that their statistics are not stationary over the image. [\[link\]](#) demonstrates this by splitting the 'Lenna' image into 64 blocks of 32×32 pels, and calculating the Fourier log power spectrum of each block. The wide variation in spectra is clearly seen. Blocks with dominant edge directions produce spectra with lines normal to the edges, and those containing the feathers of the hat generate a broad spread of energy at all frequencies. However a bright centre, indicating dominant low frequency components, is common to all blocks.



Fourier log power spectra of 'Lenna' image split into 64 blocks of 32×32 pels.

We conclude that in many regions of a typical image, most of the signal energy is contained in a relatively small number of spectral components, many of which are at low frequencies. However, between regions, the location of the main components changes significantly.

The concentration of spectral energy is the key to compression. If a signal can be reconstructed from its Fourier transform, and many of the transform coefficients are very small, then a close approximation to the original can be reconstructed from just the larger transform coefficients, so only these coefficients need be transmitted.

In practice, the Fourier transform is not very suitable for compression because it generates complex coefficients and it is badly affected by discontinuities at block boundaries (half-sine windowing was used in [\[link\]](#) and [\[link\]](#) to reduce boundary effects but this would prevent proper reconstruction of the image). In [further discussion](#), we demonstrate the principles of image compression using the Haar transform, perhaps the simplest of all transforms.

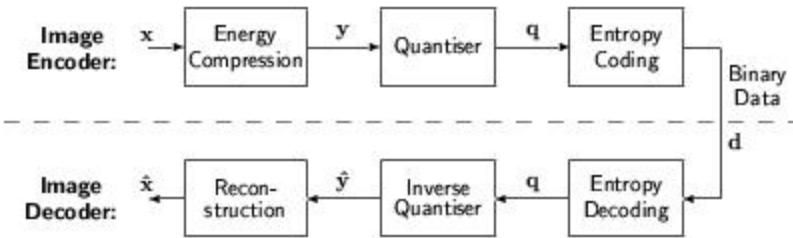
A Basic Image Compression Example

We shall represent a monochrome (luminance) image by a matrix x whose elements are $x(\mathbf{n})$, where $\mathbf{n} = (n_1 \ n_2)$ is the integer vector of row and column indexes. The energy of x is defined as

Equation:

$$\text{Energy of } x = \sum_{\mathbf{n}} x^2(\mathbf{n})$$

where the sum is performed over all \mathbf{n} in x .



The basic block diagram of an image coding system.

[\[link\]](#) shows the main blocks in any image coding system. The decoder is the inverse of the encoder. The three encoder blocks perform the following tasks:

- **Energy compression** - This is usually a transformation or filtering process which aims to concentrate a high proportion of the energy of the image x into as few samples (coefficients) of y as possible while preserving the total energy of x in y . This minimises the number of non-zero samples of y which need to be transmitted for a given level of distortion in the reconstructed image x .
- **Quantisation** - This represents the samples of y to a given level of accuracy in the integer matrix q . The quantiser step size controls the tradeoff between distortion and bit rate and may be adapted to take

account of human visual sensitivities. The inverse quantiser reconstructs \hat{y} , the best estimate of y from q .

- **Entropy coding** - This encodes the integers in q into a serial bit stream d , using variable-length entropy codes which attempt to minimise the total number of bits in d , based on the statistics (PDFs) of various classes of samples in q .

The energy compression / reconstruction and the entropy coding / decoding processes are normally all lossless. Only the quantiser introduces loss and distortion: \hat{y} is a distorted version of y , and hence x is a distorted version of x . In the absence of quantisation, if $\hat{y} = y$, then $x = x$.

The Haar Transform

Probably the simplest useful energy compression process is the Haar transform. In 1-dimension, this transforms a 2-element vector $(x(1) \ x(2))^T$ into $(y(1) \ y(2))^T$ using:

Equation:

$$\begin{pmatrix} y(1) \\ y(2) \end{pmatrix} = T \begin{pmatrix} x(1) \\ x(2) \end{pmatrix}$$

where $T = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$. Thus $y(1)$ and $y(2)$ are simply the sum and difference of $x(1)$ and $x(2)$, scaled by $\frac{1}{\sqrt{2}}$ to preserve energy.

Note that T is an orthonormal matrix because its rows are orthogonal to each other (their dot products are zero) and they are normalised to unit magnitude. Therefore $T^{-1} = T^T$. (In this case T is symmetric so $T^T = T$.) Hence we may recover x from y using:

Equation:

$$\begin{pmatrix} x(1) \\ x(2) \end{pmatrix} = T^T \begin{pmatrix} y(1) \\ y(2) \end{pmatrix}$$

In 2-dimensions x and y become 2×2 matrices. We may transform first the columns of x , by premultiplying by T , and then the rows of the result by postmultiplying by T^T . Hence:

Equation:

$$y = T x T^T$$

and to invert:

Equation:

$$x = T^T y T$$

To show more clearly what is happening:

If

$$x = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

then

$$y = \frac{1}{2} \begin{pmatrix} a + b + c + d & a - b + c - d \\ a + b - c - d & a - b - c + d \end{pmatrix}$$

These operations correspond to the following filtering processes:

- **Top left:** $a + b + c + d$ = 4-point average or 2-D lowpass (Lo-Lo) filter.
- **Top right:** $a - b + c - d$ = Average horizontal gradient or horizontal highpass and vertical lowpass (Hi-Lo) filter.
- **Lower left:** $a + b - c - d$ = Average vertical gradient or horizontal lowpass and vertical highpass (Lo-Hi) filter.
- **Lower right:** $a - b - c + d$ = Diagonal curvature or 2-D highpass (Hi-Hi) filter.

To apply this transform to a complete image, we group the pels into 2×2 blocks and apply [\[link\]](#) to each block. The result (after reordering) is shown in [\[link\]](#). To view the result sensibly, we have grouped all the top left components of the 2×2 blocks in y together to form the top left subimage in [\[link\]](#), and done the same for the components in the other 3 positions to form the corresponding other 3 subimages.



Original [\[link\]](#) and the Level 1 Haar transform [\[link\]](#) of the 'Lenna' image.

It is clear from [\[link\]](#) that most of the energy is contained in the top left (Lo-Lo) subimage and the least energy is in the lower right (Hi-Hi) subimage. Note how the top right (Hi-Lo) subimage contains the near-vertical edges and the lower left (Lo-Hi) subimage contains the near-horizontal edges.

The energies of the subimages and their percentages of the total are:

Lo-Lo	Hi-Lo	Lo-Hi	Hi-Hi
201.73×10^6	4.56×10^6	1.89×10^6	0.82×10^6
96.5%	2.2%	0.9%	0.4%

Total energy in [\[link\]](#) and [\[link\]](#) = 208.99×10^6 .

We see that a significant compression of energy into the Lo-Lo subimage has been achieved. However the energy measurements do not tell us directly how much data compression this gives.

A much more useful measure than energy is the **entropy** of the subimages after a given amount of quantisation. This gives the minimum number of bits per pel needed to represent the quantised data for each subimage, to a given accuracy, assuming that we use an ideal entropy code. By comparing the total entropy of the 4 subimages with that of the original image, we can estimate the compression that one level of the Haar transform can provide.

Entropy

Entropy of source information was discussed in the third-year E5 Information and Coding course. For an image x , quantised to M levels, the entropy H_x is defined as:

Equation:

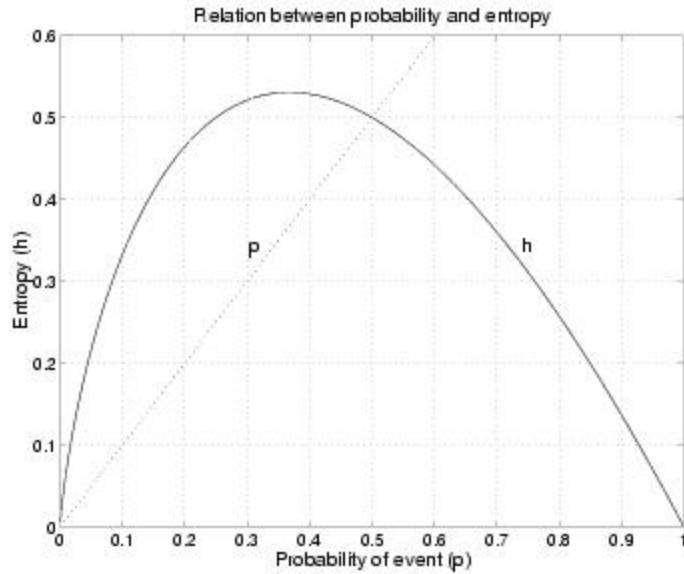
$$\begin{aligned} H_x &= \sum_{i=0}^{M-1} p_i \log_2 \frac{1}{p_i} \\ &= - \sum_{i=0}^{M-1} p_i \log_2 p_i \end{aligned}$$

where p_i , $i = 0$ to $M - 1$, is the probability of the i^{th} quantiser level being used (often obtained from a histogram of the pel intensities).

H_x represents the mean number of bits per pel with which the quantised image x can be represented using an ideal variable-length entropy code. A Huffman code usually approximates this bit-rate quite closely.

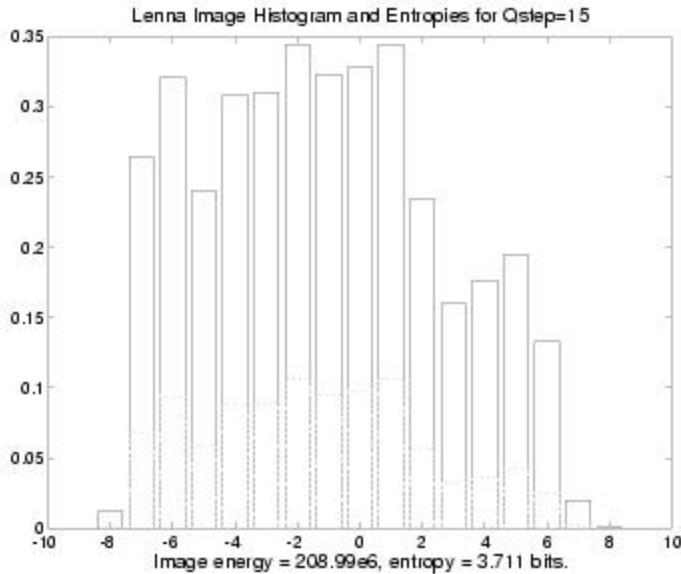
To obtain the number of bits to code an image (or subimage) x containing N pels:

- A histogram of x is measured using M bins corresponding to the M quantiser levels.
- The M histogram counts are each divided by N to give the probabilities p_i , which are then converted into entropies $h_i = - (p_i \log_2 p_i)$. This conversion law is illustrated in [\[link\]](#) and shows that probabilities close to zero or one produce low entropy and intermediate values produce entropies near 0.5.
- The entropies h_i of the separate quantiser levels are summed to give the total entropy H_x for the subimage.
- Multiplying H_x by N gives the estimated total number of bits needed to code x , assuming an ideal entropy code is available which is matched to the histogram of x .

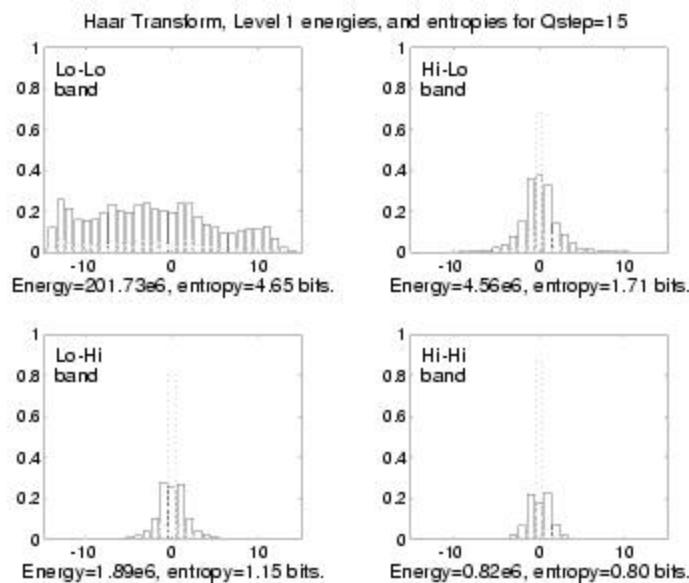


Conversion from probability p_i to entropy $h_i = -(p_i \log_2 p_i)$.

[\[link\]](#) shows the probabilities p_i and entropies h_i for the original Lenna image and [\[link\]](#) shows these for each of the subimages in this previous [figure](#), assuming a uniform quantiser with a step-size $Q_{\text{step}} = 15$ in each case. The original Lenna image contained pel values from 3 to 238 and a mean level of 120 was subtracted from each pel value before the image was analysed or transformed in order that all samples would be approximately evenly distributed about zero (a natural feature of highpass subimages).



Probability histogram (dashed) and entropies (solid) of the Lenna image in [original image](#).



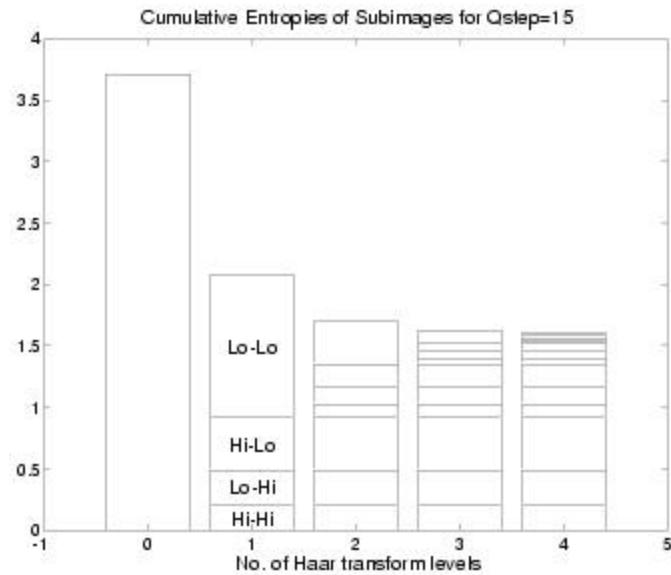
Probability histogram (dashed) and entropies (solid) of the four subimages of the Level 1 Haar transform of Lenna (see previous [figure](#)).

The Haar transform preserves energy and so the expected distortion energy from quantising the transformed image y with a given step size Q_{step} will be approximately the same as that from quantising the input image x with the same step size. This is because quantising errors can usually be modeled as independent random processes with variance (energy) $= \frac{Q_{\text{step}}^2}{12}$ and the total squared quantising error (distortion) will tend to the sum of the variances over all pels. This applies whether the error energies are summed before or after the inverse transform (reconstruction) in the decoder.

Hence equal quantiser step sizes before and after an energy-preserving transformation should generate equivalent quantising distortions and provide a fair estimate of the compression achieved by the transformation.

The first two columns of [\[link\]](#) (original and level 1) compare the entropy (mean bit rate) per pel for the original image (3.71 bit / pel) with that of the Haar transformed image of this previous [figure](#) (2.08 bit / pel), using $Q_{\text{step}} = 15$. Notice that the entropy of the original image is almost as great as the 4 bit / pel that would be needed to code the 16 levels using a simple fixed-length code, because the histogram is relatively uniform.

The level 1 column of [\[link\]](#) shows the contribution of each of the subimages of this previous [figure](#) to the total entropy per pel (the entropies from [\[link\]](#) have been divided by 4 since each subimage has one quarter of the total number of pels). the Lo-Lo subimage contributes 56% to the total entropy (bit rate) and has similar spatial correlations to the original image. Hence it is a logical step to apply the Haar transform again to this subimage.



Mean bit rate for the original Lenna image and for the Haar transforms of the image after 1 to 4 levels, using a quantiser step size $Q_{\text{step}} = 15$.

The Multi-level Haar Transform

(a) of [\[link\]](#) shows the result of applying the Haar transform to the Lo-Lo subimage of this previous [figure](#) and [\[link\]](#) shows the probabilities p_i and entropies h_i for the 4 new subimages.

The level 2 column of the figure [Cumulative Entropies of Subimages for Qstep=15](#) shows how the total bit rate can be reduced by transforming the level 1 Lo-Lo subimage into four level 2 subimages. The process can be repeated by transforming the final Lo-Lo subimage again and again, giving the subimages in (b) of [\[link\]](#) and (c) of [\[link\]](#) and the histograms in [\[link\]](#) and [\[link\]](#). The levels 3 and 4 columns of the figure [Cumulative Entropies of Subimages for Qstep=15](#) show that little is gained by transforming to more than 4 levels.

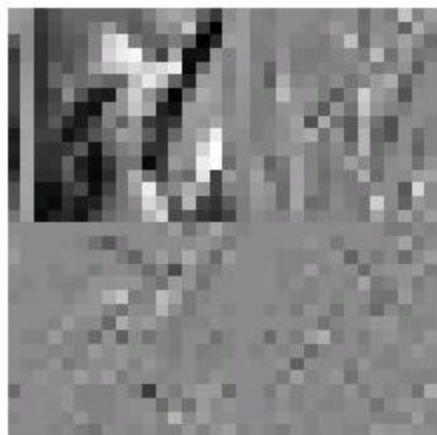
However a total compression ratio of 4 bit/pel : 1.61 bit/pel = 2.45 : 1 has been achieved (in theory).



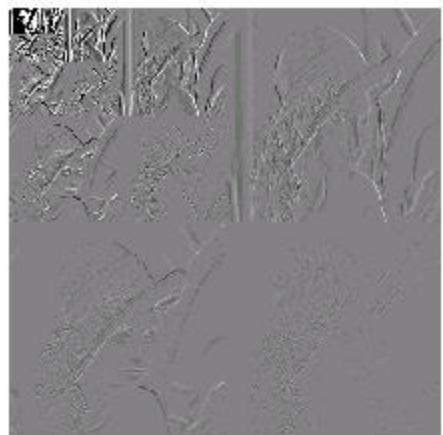
(a)



(b)

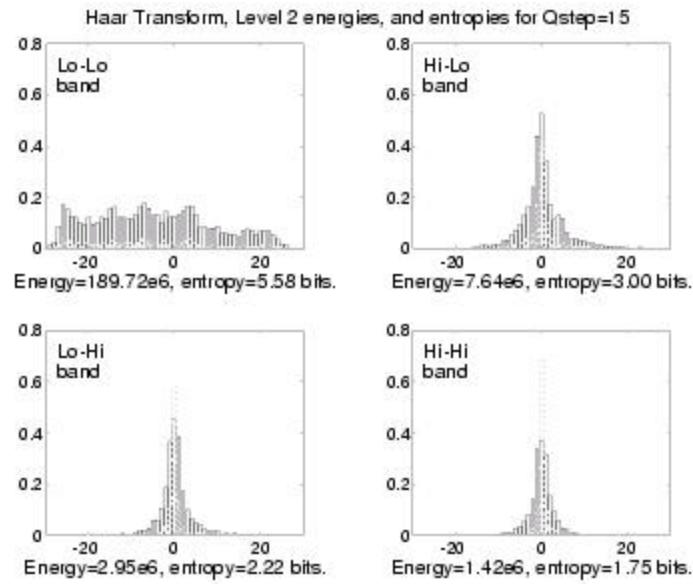


(c)

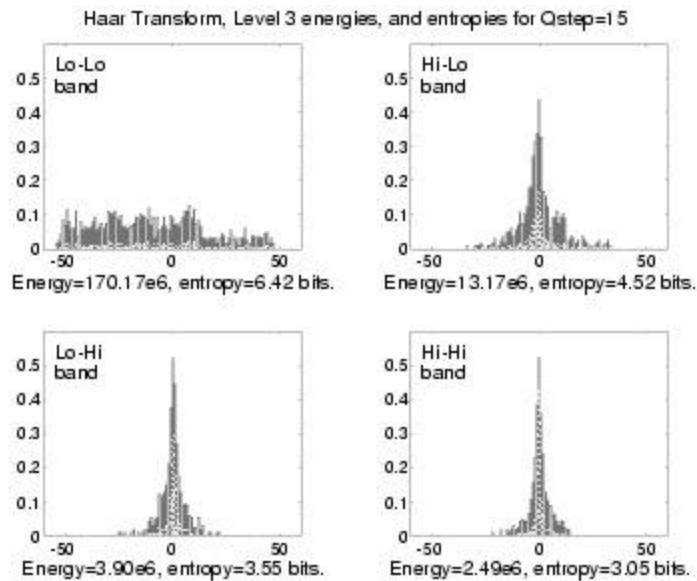


(d)

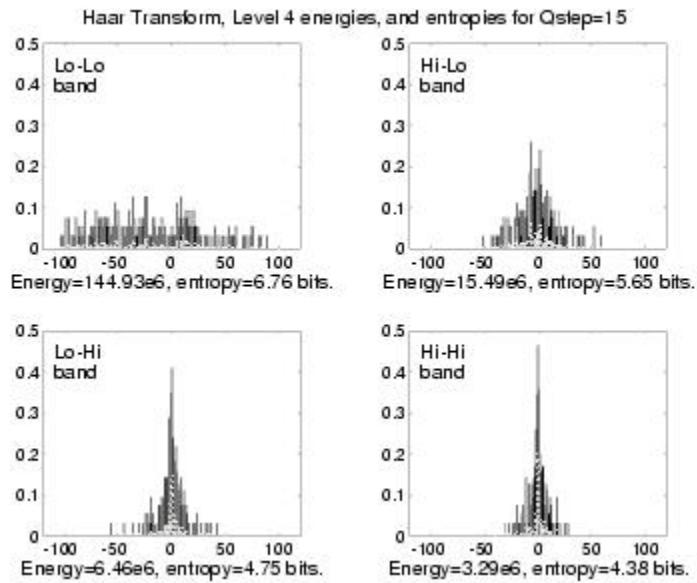
Levels 2(a), 3(b), and 4(c) Haar transforms of Lenna;
and at all of levels 1 to 4(d).



The probabilities p_i and entropies h_i
for the 4 subimages at level 2.



The probabilities p_i and entropies h_i
for the 4 subimages at level 3.



The probabilities p_i and entropies h_i
for the 4 subimages at level 4.



(a)

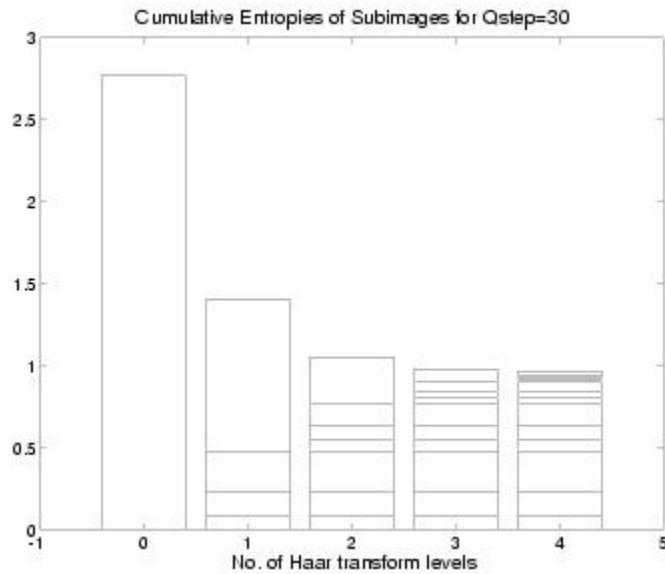


(b)

Images reconstructed from (a) the original Lenna, and
(b) the 4-level Haar transform, each quantised with
 $Q_{\text{step}} = 15$. The rms error of (a) = 4.3513, and of (b) =
3.5343.

Note the following features of the 4-level Haar transform:

- (d) of [\[link\]](#) shows the subimages from all 4 levels of the transform and illustrates the transform's **multi-scale** nature. It also shows that all the subimages occupy the same total area as the original and hence that the total number of transform output samples (coefficients) equals the number of input pels - there is **no redundancy**.
- From the Lo-Lo subimage histograms of the figure [Haar Transform, Level 1 energies, and entropies for Qstep=15](#), [\[link\]](#), [\[link\]](#) and [\[link\]](#), we see the magnitudes of the Lo-Lo subimage samples increasing with transform level. This is because energy is being conserved and most of it is being concentrated in fewer and fewer Lo-Lo samples. (The DC gain of the Lo-Lo filter of this previous [equation](#) is 2.)
- We may reconstruct the image from the transform samples ((d) of [\[link\]](#)), quantised to $Q_{\text{step}} = 15$, by inverting the transform, using the right hand part of this [equation](#). We then get the image in (b) of [\[link\]](#). Contrast this with (a) of [\[link\]](#), obtained by quantising the pels of the original directly to $Q_{\text{step}} = 15$, in which contour artifacts are much more visible. Thus the transform provides improved subjective quality as well as significant data compression. The improved quality arises mainly from the high amplitude of the low frequency transform samples, which means that they are quantised to many more levels than the basic pels would be for a given Q_{step} .
- If Q_{step} is doubled to 30, then the entropies of all the subimages are reduced as shown in [\[link\]](#) (compare this with the figure, [Cumulative Entropies of Subimages for Qstep=15](#) in which $Q_{\text{step}} = 15$). The mean bit rate with the 4-level Haar transform drops from 1.61 to 0.97 bit/pel. However the reconstructed image quality drops to that shown in (b) of [\[link\]](#). For comparison, (a) of [\[link\]](#) shows the quality if x is directly quantised with $Q_{\text{step}} = 30$.



Mean bit rate for the original Lenna image and for the Haar transforms of the image after 1 to 4 levels, using a quantiser step size $Q_{\text{step}} = 30$.



(a)



(b)

Images reconstructed from (a) the original Lenna, and (b) the 4-level Haar transform, each quantised with $Q_{\text{step}} = 30$. The rms error of (a) = 8.6219, and of (b) = 5.8781.

Use of Laplacian PDFs in Image Compression

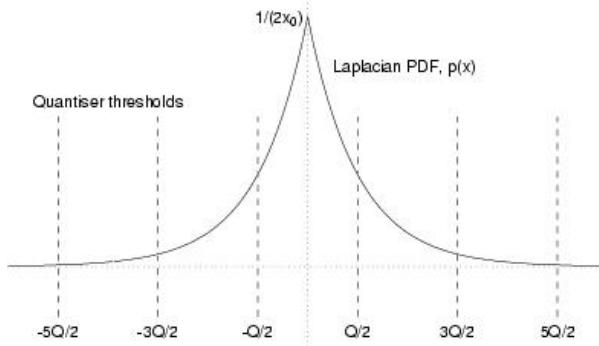
It is found to be appropriate and convenient to model the distribution of many types of transformed image coefficients by Laplacian distributions. It is appropriate because much real data is approximately modeled by the Laplacian probability density function (PDF), and it is convenient because the mathematical form of the Laplacian PDF is simple enough to allow some useful analytical results to be derived.

A Laplacian PDF is a back-to-back pair of exponential decays and is given by:

Equation:

$$p(x) = \frac{1}{2x_0} e^{-\frac{|x|}{x_0}}$$

where x_0 is the equivalent of a **time constant** which defines the **width** of the PDF from the centre to the $\frac{1}{e}$ points. The initial scaling factor ensures that the area under $p(x)$ is unity, so that it is a valid PDF. [\[link\]](#) shows the shape of $p(x)$.



Laplacian PDF, $p(x)$, and typical quantiser decision thresholds, shown for the case when the quantiser step size $Q = 2x_0$

The mean of this PDF is zero and the variance is given by:

Equation:

$$\begin{aligned} v(x_0) &= \int_{-\infty}^{\infty} x^2 p(x) \, dx \\ &= 2 \int \frac{x^2}{2x_0} e^{-\frac{|x|}{x_0}} \, dx \\ &= 2x_0^2 \end{aligned}$$

(using integration by parts twice).

Hence the standard deviation is:

Equation:

$$\begin{aligned}\sigma(x_0) &= \sqrt{v(x_0)} \\ &= \sqrt{2}x_0\end{aligned}$$

Given the variance (power) of a subimage of transformed pels, we may calculate x_0 and hence determine the PDF of the subimage, assuming a Laplacian shape. We now show that, if we quantise the subimage using a uniform quantiser with step size Q , we can calculate the entropy of the quantised samples and thus estimate the bit rate needed to encode the subimage in bits/pel. This is a powerful analytical tool as it shows how the compressed bit rate relates directly to the energy of a subimage. The vertical dashed lines in [\[link\]](#) show the decision thresholds for a typical quantiser for the case when $Q = 2x_0$.

First we analyse the probability of a pel being quantised to each step of the quantiser. This is given by the area under $p(x)$ between each adjacent pair of quantiser thresholds.

- Probability of being at step 0, $p_0 = \Pr[-(\frac{1}{2}Q) < x < \frac{1}{2}Q] = 2\Pr[0 < x < \frac{1}{2}Q]$
- Probability of being at step k , $p_k = \Pr[(k - \frac{1}{2})Q < x < (k + \frac{1}{2})Q]$

First, for $x_2 \geq x_1 \geq 0$, we calculate:

$$\Pr[x_1 < x < x_2] = \int_{x_1}^{x_2} p(x) \, dx = \left(-\frac{1}{2} \right) e^{-\frac{x}{x_0}} \Big|_{x_1}^{x_2} = \frac{1}{2} \left(e^{-\frac{x_1}{x_0}} - e^{-\frac{x_2}{x_0}} \right)$$

Therefore,

Equation:

$$p_0 = 1 - e^{-\frac{Q}{2x_0}}$$

and, for $k \geq 1$,

Equation:

$$\begin{aligned}p_k &= \frac{1}{2} \left(e^{-\frac{(k-\frac{1}{2})Q}{x_0}} - e^{-\frac{(k+\frac{1}{2})Q}{x_0}} \right) \\ &= \sinh\left(\frac{Q}{2x_0}\right) e^{-\frac{kQ}{x_0}}\end{aligned}$$

By symmetry, if k is nonzero, $p_{-k} = p_k = \sinh\left(\frac{Q}{2x_0}\right) e^{-\frac{|k|Q}{x_0}}$

Now we can calculate the entropy of the subimage:

Equation:

$$\begin{aligned}H &= -\sum_{k=-\infty}^{\infty} p_k \log_2 p_k \\ &= (- (p_0 \log_2 p_0)) - 2 \sum_{k=1}^{\infty} p_k \log_2 p_k\end{aligned}$$

To make the evaluation of the summation easier when we substitute for p_k , we let

$$p_k = \alpha r^k$$

where $\alpha = \sinh\left(\frac{Q}{2x_0}\right)$ and $r = e^{-\frac{Q}{x_0}}$. Therefore,

Equation:

$$\begin{aligned}\sum_{k=1}^{\infty} p_k \log_2 p_k &= \sum_{k=1}^{\infty} \alpha r^k \log_2 (\alpha r^k) \\ &= \sum_{k=1}^{\infty} \alpha r^k (\log_2 \alpha + k \log_2 r) \\ &= \alpha \log_2 \alpha \sum_{k=1}^{\infty} r^k + \alpha \log_2 r \sum_{k=1}^{\infty} k r^k\end{aligned}$$

Now $\sum_{k=1}^{\infty} r^k = \frac{r}{1-r}$ and, differentiating by r : $\sum_{k=1}^{\infty} k r^{k-1} = \frac{1}{(1-r)^2}$. Therefore,

Equation:

$$\begin{aligned}\sum_{k=1}^{\infty} p_k \log_2 p_k &= \alpha \log_2 \alpha \frac{r}{1-r} + \alpha \log_2 r \frac{r}{(1-r)^2} \\ &= \frac{\alpha r}{1-r} \left(\log_2 \alpha + \frac{\log_2 r}{1-r} \right)\end{aligned}$$

and

Equation:

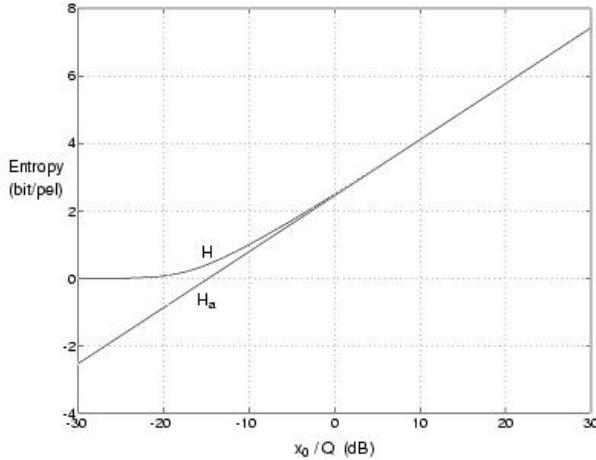
$$p_0 \log_2 p_0 = (1 - \sqrt{r}) \log_2 (1 - \sqrt{r})$$

Hence the entropy is given by:

Equation:

$$H = \left(-((1 - \sqrt{r}) \log_2 (1 - \sqrt{r})) - \frac{2\alpha r}{1-r} \left(\log_2 \alpha + \frac{\log_2 r}{1-r} \right) \right)$$

Because both α and r are functions of $\frac{Q}{x_0}$, then H is a function of just $\frac{Q}{x_0}$ too. We expect that, for constant Q , as the energy of the subimage increases, the entropy will also increase approximately logarithmically, so we plot H against $\frac{x_0}{Q}$ in dB in [\[link\]](#). This shows that our expectations are born out.



Entropy H and approximate entropy H_a of a quantised subimage with Laplacian PDF, as a function of $\frac{x_0}{Q}$ in dB.

We can show this in theory by considering the case when $\frac{x_0}{Q} \gg 1$, when we find that:

$$\alpha \simeq \frac{Q}{2x_0}$$

$$r \simeq 1 - \frac{Q}{x_0} \simeq 1 - 2\alpha$$

$$\sqrt{r} \simeq 1 - \alpha$$

Using the approximation $\log_2(1 - \varepsilon) \simeq -\frac{\varepsilon}{\ln(2)}$ for small ε , it is then fairly straightforward to show that

$$H \simeq -\log_2 \alpha + \frac{1}{\ln(2)} \simeq \log_2 \frac{2ex_0}{Q}$$

We denote this approximation as H_a in [\[link\]](#), which shows how close to H the approximation is, for $x_0 > Q$ (i.e. for $\frac{x_0}{Q} > 0$ dB).

We can compare the entropies calculated using [\[link\]](#) with those that were calculated from the bandpass subimage histograms, as given in these figures describing Haar transform energies and entropies; [level 1 energies](#), [level 2 energies](#), [level 3 energies](#), and [level 4 energies](#). (The Lo-Lo subimages have PDFs which are more uniform and do not fit the Laplacian model well.) The values of x_0 are calculated from:

$$x_0 = \frac{\text{std. dev.}}{\sqrt{2}} = \sqrt{\frac{\text{subimage energy}}{2 \text{ (no of pels in subimage)}}}$$

The following table shows this comparison:

Transform level	Subimage type	Energy ($\times 10^6$)	No of pels	x_0	Laplacian entropy	Measured entropy
1	Hi-Lo	4.56	16384	11.80	2.16	1.71
1	Lo-Hi	1.89	16384	7.59	1.58	1.15
1	Hi-Hi	0.82	16384	5.09	1.08	0.80
2	Hi-Lo	7.64	4096	30.54	3.48	3.00
2	Lo-Hi	2.95	4096	18.98	2.81	2.22
2	Hi-Hi	1.42	4096	13.17	2.31	1.75
3	Hi-Lo	13.17	1024	80.19	4.86	4.52
3	Lo-Hi	3.90	1024	43.64	3.99	3.55
3	Hi-Hi	2.49	1024	34.87	3.67	3.05
4	Hi-Lo	15.49	256	173.9	5.98	5.65
4	Lo-Hi	6.46	256	112.3	5.35	4.75
4	Hi-Hi	3.29	256	80.2	4.86	4.38

We see that the entropies calculated from the energy via the Laplacian PDF method (second column from the right) are approximately 0.5 bit/pel greater than the entropies measured from the Lenna subimage histograms. This is due to the heavier tails of the actual PDFs compared with the Laplacian exponentially decreasing tails. More accurate entropies can be obtained if x_0 is obtained from the mean absolute values of the pels in each subimage. For a Laplacian PDF we can show that **Equation:**

$$\begin{aligned}
 \text{Mean absolute value} &= \int_{-\infty}^{\infty} |x| p(x) \, dx \\
 &= 2 \int_0^{\infty} \frac{x}{2x_0} e^{-\frac{x}{x_0}} \, dx \\
 &= x_0
 \end{aligned}$$

This gives values of x_0 that are about 20% lower than those calculated from the energies and the calculated entropies are then within approximately 0.2 bit/pel of the measured entropies.

Practical Entropy Coding Techniques

In the [module of Use of Laplacian PDFs in Image Compression](#) we have assumed that ideal entropy coding has been used in order to calculate the bit rates for the coded data. In practise we must use real codes and we shall now see how this affects the compression performance.

There are three main techniques for achieving entropy coding:

- **Huffman Coding** - one of the simplest variable length coding schemes.
- **Run-length Coding (RLC)** - very useful for binary data containing long runs of ones or zeros.
- **Arithmetic Coding** - a relatively new variable length coding scheme that can combine the best features of Huffman and run-length coding, and also adapt to data with non-stationary statistics.

We shall concentrate on the Huffman and RLC methods for simplicity. Interested readers may find out more about Arithmetic Coding in chapters 12 and 13 of the JPEG Book.

First we consider the change in compression performance if simple Huffman Coding is used to code the subimages of the 4-level Haar transform.

The calculation of entropy in this [equation](#) from our discussion of entropy assumed that each message with probability p_i could be represented by a word of length $i = -\log_2 p_i$ bits. Huffman codes require the i to be integers and assume that the p_i are adjusted to become:

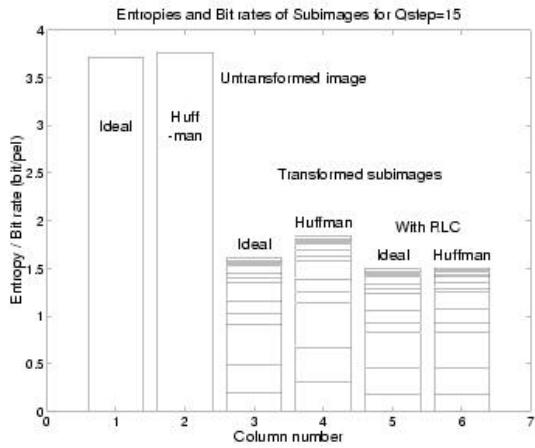
Equation:

$$\hat{p}_i = 2^{-i}$$

where the i are integers, chosen subject to the constraint that $\sum_i \hat{p}_i \leq 1$ (to guarantee that sufficient uniquely decodable code words are available) and such that the mean Huffman word length (Huffman entropy), $\hat{H} = \sum_i p_i i$, is minimised.

We can use the probability histograms which generated the entropy plots in figures of [level 1 energies](#), [level 2 energies](#), [level 3 energies](#) and [level 4 energies](#) to calculate the Huffman entropies \hat{H} for each subimage and compare these with the true entropies to see the loss in performance caused by using real Huffman codes.

An algorithm for finding the optimum codesizes i is recommended in the JPEG specification [*the JPEG Book*, Appendix A, Annex K.2, fig K.1]; and a Matlab M-file to implement it is given in [M-file code](#).



Comparison of entropies (columns 1, 3, 5) and Huffman coded bit rates (columns 2, 4, 6) for the original (columns 1 and 2) and transformed (columns 3 to 6) Lenna images. In columns 5 and 6, the zero amplitude state is run-length encoded to produce many states with probabilities < 0.5.

Column:	1	2	3	4	5	6	-
			0.0264	0.0265	0.0264	0.0266	
			0.0220	0.0222	0.0221	0.0221	Level 4
			0.0186	0.0187	0.0185	0.0186	
			0.0171	0.0172	0.0171	0.0173	-
			0.0706	0.0713	0.0701	0.0705	
			0.0556	0.0561	0.0557	0.0560	Level 3
	3.7106	3.7676	0.0476	0.0482	0.0466	0.0471	-

Column:	1	2	3	4	5	6	-
			0.1872	0.1897	0.1785	0.1796	
			0.1389	0.1413	0.1340	0.1353	Level 2
			0.1096	0.1170	0.1038	0.1048	-
			0.4269	0.4566	0.3739	0.3762	
			0.2886	0.3634	0.2691	0.2702	Level 1
			0.2012	0.3143	0.1819	0.1828	-
Totals:	3.7106	3.7676	1.6103	1.8425	1.4977	1.5071	

Numerical results used in the figure - Entropies and Bit rates of Subimages for Qstep=15

[\[link\]](#) shows the results of applying this algorithm to the probability histograms and [\[link\]](#) lists the same results numerically for ease of analysis. Columns 1 and 2 compare the ideal entropy with the mean word length or bit rate from using a Huffman code (the Huffman entropy) for the case of the untransformed image where the original pels are quantized with $Q_{\text{step}} = 15$. We see that the increase in bit rate from using the real code is:

$$\frac{3.7676}{3.7106} - 1 = 1.5\%$$

But when we do the same for the 4-level transformed subimages, we get columns 3 and 4. Here we see that real Huffman codes require an increase in bit rate of:

$$\frac{1.8425}{1.6103} - 1 = 14.4\%$$

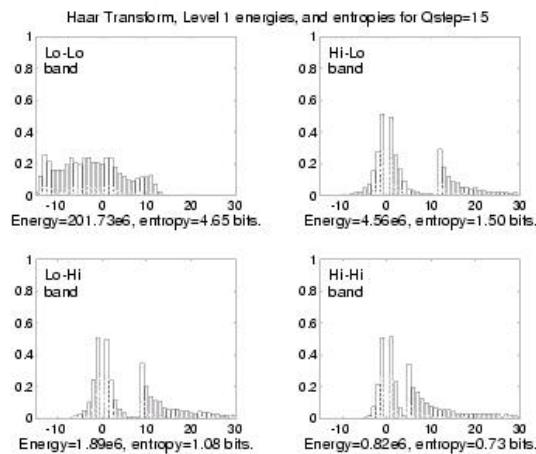
Comparing the results for each subimage in columns 3 and 4, we see that most of the increase in bit rate arises in the three level-1 subimages at the bottom of the columns. This is because each of the probability histograms for these subimages (see [figure](#)) contain one probability that is greater than 0.5. Huffman codes cannot allocate a word length of less than 1 bit to a given event, and so they start to lose efficiency rapidly when $-\log_2 p_i$ becomes less than 1, ie when $p_i > 0.5$.

Run-length codes (RLCs) are a simple and effective way of improving the efficiency of Huffman coding when one event is much more probable than all of the others combined. They operate as follows:

- The pels of the subimage are scanned sequentially (usually in columns or rows) to form a long 1-dimensional vector.

- Each run of consecutive zero samples (the most probable events) in the vector is coded as a single event.
- Each non-zero sample is coded as a single event in the normal way.
- The two types of event (runs-of-zeros and non-zero samples) are allocated separate sets of codewords in the same Huffman code, which may be designed from a histogram showing the frequencies of all events.
- To limit the number of run events, the maximum run length may be limited to a certain value (we have used 128) and runs longer than this may be represented by two or more run codes in sequence, with negligible loss of efficiency.

Hence RLC may be added before Huffman coding as an extra processing step, which converts the most probable event into many separate events, each of which has $p_i < 0.5$ and may therefore be coded efficiently. [\[link\]](#) shows the new probability histograms and entropies for level 1 of the Haar transform when RLC is applied to the zero event of the three bandpass subimages. Comparing this with a [previous figure](#), note the absence of the high probability zero events and the new states to the right of the original histograms corresponding to the run lengths.



Probability histograms (dashed) and entropies (solid) of the four subimage of the Level 1 Haar transform of Lenna (see [figure](#)) after RLC.

The total entropy **per event** for an RLC subimage is calculated as before from the entropy histogram. However to get the entropy **per pel** we scale the entropy by the ratio of the number of events (runs and non-zero samples) in the subimage to the number of pels in the subimage (note that with RLC this ratio will no longer equal one - it will hopefully be much less).

[\[link\]](#) gives the entropies per pel after RLC for each subimage, which are now **less** than the entropies in this [figure](#). This is because RLC takes advantage of spatial clustering of the zero samples in a subimage, rather than just depending on the histogram of amplitudes.

Clearly if all the zeros were clustered into a single run, this could be coded much more efficiently than if they are distributed into many runs. The entropy of the zero event tells us the mean number

of bits to code each zero pel **if the zero pels are distributed randomly**, ie if the probability of a given pel being zero does not depend on the amplitudes of any nearby pels.

In typical bandpass subimages, non-zero samples tend to be clustered around key features such as object boundaries and areas of high texture. Hence RLC usually reduces the entropy of the data to be coded. There are many other ways to take advantage of clustering (correlation) of the data - RLC is just one of the simplest.

In [\[link\]](#), comparing column 5 with column 3, we see the modest (7%) reduction in entropy per pel achieved by RLC, due clustering in the Lenna image. The main advantage of RLC is apparent in column 6, which shows the mean bit rate per pel when we use a real Huffman code on the RLC histograms of [\[link\]](#). The increase in bit rate over the RLC entropy is only

$$\frac{1.5071}{1.4977} - 1 = 0.63\%$$

compared with 14.4% when RLC is not used (columns 3 and 4).

Finally, comparing column 6 with column 3, we see that, relative to the simple entropy measure, combined RLC and Huffman coding can **reduce** the bit rate by

$$1 - \frac{1.5071}{1.6103} = 6.4\%$$

The closeness of this ratio to unity justifies our use of simple entropy as a tool for assessing the information compression properties of the Haar transform - and of other energy compression techniques as we meet them.

The following is the listing of the M-file to calculate the Huffman entropy from a given histogram.

```
% Find Huffman code sizes: JPEG fig K.1, procedure
Code_size.
% huffhist contains the histogram of event counts
(frequencies).
freq = huffhist(:);
codesize = zeros(size(freq));
others = -ones(size(freq)); %Pointers to next symbols in
code tree.

% Find non-zero entries in freq, and loop until only 1 entry
left.
nz = find(freq > 0);
while length(nz) > 1,
% Find v1 for least value of freq(v1) > 0.
[y,i] = min(freq(nz));
v1 = nz(i);
```

```

% Find v2 for next least value of freq(v2) > 0.
nz = nz([1:(i-1) (i+1):length(nz)]); % Remove v1 from nz.
[y,i] = min(freq(nz));
v2 = nz(i);
% Combine frequency values.
freq(v1) = freq(v1) + freq(v2);
freq(v2) = 0;
codesize(v1) = codesize(v1) + 1;
% Increment code sizes for all codewords in this tree
branch.
while others(v1) > -1,
    v1 = others(v1);
    codesize(v1) = codesize(v1) + 1;
end
others(v1) = v2;
codesize(v2) = codesize(v2) + 1;
while others(v2) > -1,
    v2 = others(v2);
    codesize(v2) = codesize(v2) + 1;
end
nz = find(freq > 0);
end

% Generate Huffman entropies by multiplying probabilities by
code sizes.
huffent = (huffhist(:)/sum(huffhist(:))) .* codesize;

```

The Discrete Cosine Transform (DCT)

The main standard for image compression in current use is the JPEG (Joint Picture Experts Group) standard, devised and refined over the period 1985 to 1993. It is formally known as ISO Draft International standard 10981-1 and CCITT Recommendation T.81, and is described in depth in The JPEG Book by W B Pennebaker and J L Mitchell, Van Nostrand Reinhold 1993.

We shall briefly outline the baseline version of JPEG but first we consider its energy compression technique - the discrete cosine transform (DCT).

The Discrete Cosine Transform (DCT)

In [this equation](#) from our discussion of the Haar transform, we met the 2-point Haar transform and in [this equation](#) we saw that it can be easily inverted if the transform matrix T is orthonormal so that $T^{-1} = T^T$.

If T is of size $n \times n$, where $n = 2^m$, then we may easily generate larger orthonormal matrices, which lead to definitions of larger transforms.

An n -point transform is defined as:

Equation:

$$\begin{pmatrix} y(1) \\ \vdots \\ y(n) \end{pmatrix} = T \begin{pmatrix} x(1) \\ \vdots \\ x(n) \end{pmatrix}$$

where $T = \begin{pmatrix} t_{1,1} & \dots & t_{1,n} \\ \vdots & \vdots & \vdots \\ t_{n,1} & \dots & t_{n,n} \end{pmatrix}$

A 4-point orthonormal transform matrix that is equivalent to 2 levels of the Haar transform is:

Equation:

$$\begin{aligned} T &= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & \sqrt{2} & 0 & 0 \\ 0 & 0 & 0 & \sqrt{2} \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix} \\ &= \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{pmatrix} \end{aligned}$$

where $\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & \sqrt{2} & 0 & 0 \\ 0 & 0 & 0 & \sqrt{2} \end{pmatrix}$ is Haar level 2 and $\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix}$ is Haar level 1.

Similarly 3 and 4 level Haar transforms may be expressed using 8 and 16 point transform matrices respectively.

However for $n > 2$, there are better matrices than those based on the Haar transform, where **better** means **with improved energy compression properties for typical images**.

Discrete Cosine Transforms (DCTs) have some of these improved properties and are also simple to define and implement. The n rows of an n -point DCT matrix T are defined by:

$$\forall i = 1 \rightarrow n : \left(t_{1,i} = \sqrt{\frac{1}{n}} \right)$$

Equation:

$$\forall (i = 1 \rightarrow n) \wedge (k = 2 \rightarrow n) : \left(t_{k,i} = \sqrt{\frac{2}{n}} \cos\left(\frac{\pi (2i-1)(k-1)}{2n}\right) \right)$$

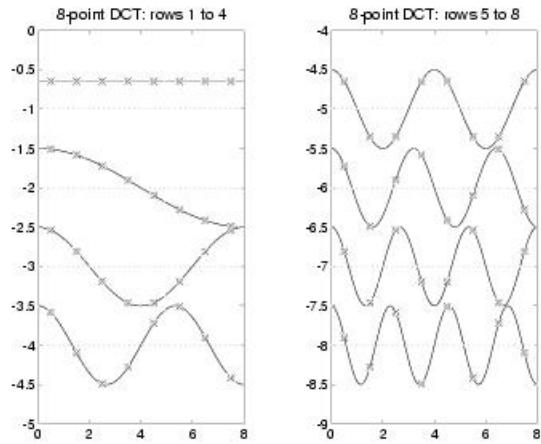
It is straightforward to show that this matrix is orthonormal for n even, since the norm of each row is unity and the dot product of any pair of rows is zero (the product terms may be expressed as the sum of a pair of cosine functions, which are each zero mean).

The 8-point DCT matrix ($n = 8$) is:

Equation:

$$T = \begin{pmatrix} 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 \\ 0.4904 & 0.4157 & 0.2778 & 0.0975 & -0.0975 & -0.2778 & -0.4157 & -0.4904 \\ 0.4619 & 0.1913 & -0.1913 & -0.4619 & -0.4619 & -0.1913 & 0.1913 & 0.4619 \\ 0.4157 & -0.0975 & -0.4904 & -0.2778 & 0.2778 & 0.4904 & 0.0975 & -0.4157 \\ 0.3536 & -0.3536 & -0.3536 & 0.3536 & 0.3536 & -0.3536 & -0.3536 & 0.3536 \\ 0.2778 & -0.4904 & 0.0975 & 0.4157 & -0.4157 & -0.0975 & 0.4904 & -0.2778 \\ 0.1913 & -0.4619 & 0.4619 & -0.1913 & -0.1913 & 0.4619 & -0.4619 & 0.1913 \\ 0.0975 & -0.2778 & 0.4157 & -0.4904 & 0.4904 & -0.4157 & 0.2778 & -0.0975 \end{pmatrix}$$

The rows of T , known as basis function, are plotted as asterisks in [\[link\]](#). The asterisks are superimposed on the underlying continuous cosine functions, used in all sizes of DCT. Only the amplitude scaling and the maximum frequency vary with the size n .



The 8-point DCT basis functions(*) and their underlying continuous cosine waves.

When we take the transform of an n -point vector using $\mathbf{y} = T\mathbf{x}$, \mathbf{x} is decomposed into a linear combination of the basis function (rows) of T , whose coefficients are the samples of \mathbf{y} , because $\mathbf{x} = T^T\mathbf{y}$.

The basis functions may also be viewed as the impulse responses of FIR filters, being applied to the data \mathbf{x} .

The DCT is closely related to the discrete Fourier transform (DFT). It represents the result of applying the $2n$ -point DFT to a vector:

$$\mathbf{x}_{2n} = \begin{pmatrix} \mathbf{x} \\ \mathbf{x}_{\text{rev}} \end{pmatrix}$$

where $\mathbf{x}_{\text{rev}} = \begin{pmatrix} x(n) \\ \dots \\ x(1) \end{pmatrix}$. \mathbf{x}_{2n} is symmetric about its centre and so the $2n$ Fourier coefficients are all purely real and symmetric about zero frequency. The n DCT coefficients are then the first n Fourier coefficients.

Note: The DFT must be defined with a half sample period offset on the indexing of the input samples for the above to be strictly true.

Standards

The 8-point DCT is the basis of the JPEG standard, as well as several other standards such as MPEG-1 and MPEG-2 (for TV and video) and H.263 (for video-phones). Hence we shall concentrate on it as our main example, but bear in mind that DCTs may be defined for a wide range of sizes n .

Fast Algorithms for the DCT

The basic n -point DCT requires n^2 multiplications and $(n^2 - n)$ additions to calculate $\mathbf{DCT}_n(\mathbf{x})$ (64 mults and 56 adds for $\mathbf{DCT}_8(\mathbf{x})$).

From the [figure](#) in our discussion of DCT, it is clear that symmetries exist in the DCT basis functions. These can be exploited to reduce the computation load of the DCT.

All the odd rows of \mathbf{DCT}_n in this [equation](#) from our discussion of DCT possess even symmetry about their centres and all the even rows possess odd symmetry. Hence we may form:

Equation:

and then form the odd and even terms in \mathbf{DCT}_n from two \mathbf{DCT}_4 transforms:

Equation:

where $\mathbf{DCT}_n^{\text{odd}}$ and $\mathbf{DCT}_n^{\text{even}}$ are the $n \times n$ matrices formed by the left halves of the odd and even rows of \mathbf{DCT}_n .

This reduces the computation to 8 add/subtract operations for [\[link\]](#) and 16 mults and 8 adds for [\[link\]](#) - almost halving the total computation load.

The matrix $\mathbf{DCT}_n^{\text{odd}}$ cannot easily be simplified much further, but can, as it possesses the same symmetries as \mathbf{DCT}_4 (it is equivalent to a 4-point DCT matrix). Hence we may use the same technique on this matrix to reduce the 16 mults and 12 adds for this product to 4 add/subtract

operations followed by a pair of x matrix products, requiring mults and adds. Finally two of these mults may be saved since one of the x matrices is just a scaled add/subtract matrix (like the Haar transform).

The total computation load for the DCT then becomes:

- add/subtract operations;
- multiply operations.

More complicated algorithms exist (JPEG Book, sections 4.3.2 to 4.3.5) which reduce the number of multiplies further. However these all require more intermediate results to be stored. In modern DSP chips this can cost more CPU cycles than the extra multiplications which can often be done simultaneously with additions. Hence the simple approach given above is frequently optimal.

The 2-dimensional DCT

In the [equation](#) from our discussion of the Haar transform:

$$y = TxT^T$$

and to invert:

$$x = T^T y T$$

we saw how a 1-D transform could be extended to 2-D by pre- and post-multiplication of a square matrix x to give a matrix result y . Our example then used 2×2 matrices, but this technique applies to square matrices of any size.

Hence the DCT may be extended into 2-D by this method.

E.g. the 8×8 DCT transforms a subimage of 8×8 pels into a matrix of 8×8 DCT coefficients.

The 2-D basis functions, from which x may be reconstructed, are given by the n^2 separate products of the columns of T^T with the rows of T . These are shown for $n = 8$ in (a) of [\[link\]](#) as 64 subimages of size 8×8 pels.

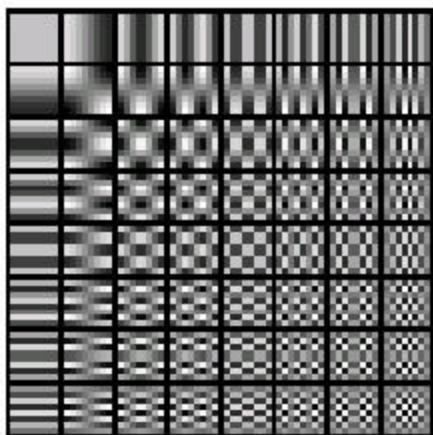
The result of applying the 8×8 DCT to the Lenna image is shown in (b) of [\[link\]](#). Here each 8×8 block of pels x is replaced by the 8×8 block of DCT coefficients y . This shows the 8×8 block structure clearly but is not very meaningful otherwise.

Part(c) of [\[link\]](#) shows the same data, reordered into 64 subimages of 32×32 coefficients each so that each subimage contains all the coefficients of a given type - e.g: the top left subimage contains all the coefficients for the top left basis function from (a) of [\[link\]](#). The other subimages and basis functions correspond in the same way.

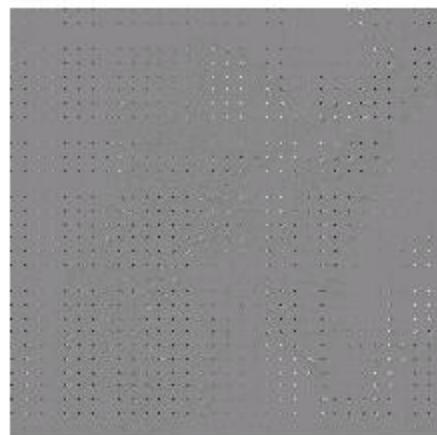
We see the major energy concentration to the subimages in the top left corner. (d) of [\[link\]](#) is an enlargement of the top left 4 subimages of (c) of [\[link\]](#) and bears a strong similarity to the group of third level Haar

subimages in (b) of [this figure](#). To emphasise this the histograms and entropies of these 4 subimages are shown in [\[link\]](#).

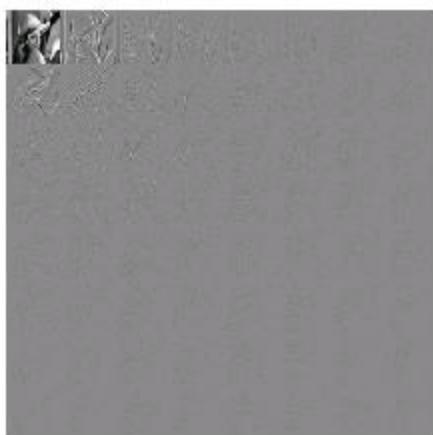
Comparing [\[link\]](#) with this [figure](#), the Haar transform equivalent, we see that the Lo-Lo bands have identical energies and entropies. This is because the basis functions are identical flat surfaces in both cases. Comparing the other 3 bands, we see that the DCT bands contain more energy and entropy than their Haar equivalents, which means **less** energy (and so hopefully less entropy) in the higher DCT bands (not shown) because the total energy is fixed (the transforms all preserve total energy). The mean entropy for all 64 subimages is 1.3622 bit/pel, which compares favourably with the 1.6103 bit/pel for the 4-level Haar transformed subimages using the same $Q_{\text{step}} = 15$.



(a)



(b)



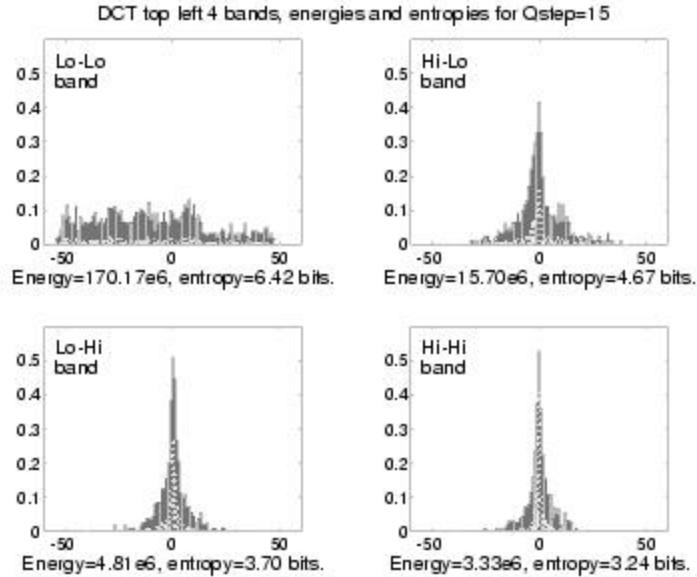
(c)



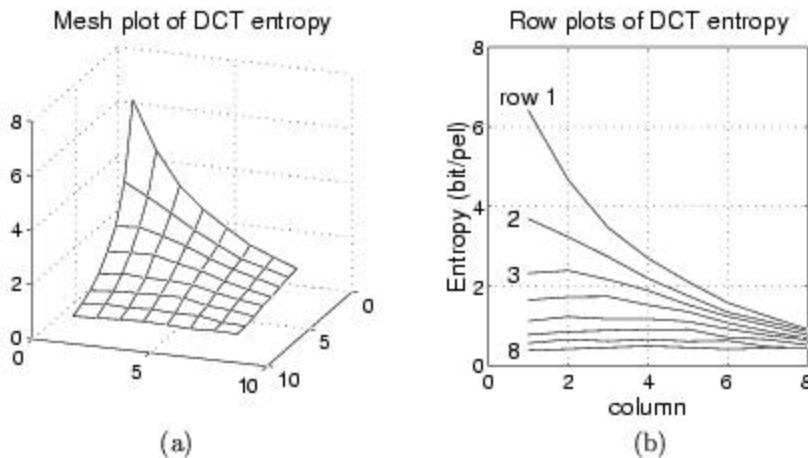
(d)

(a) Basis functions of the 8×8 DCT; (b) Lenna transformed by the 8×8 DCT; (c) reordered into

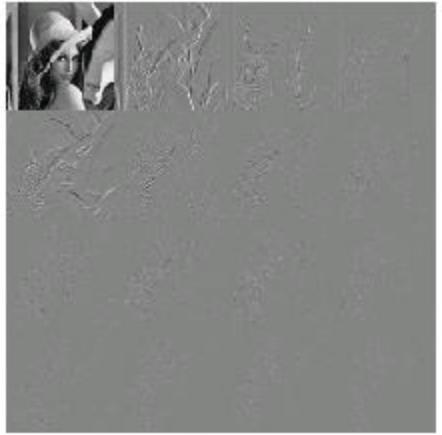
subimages grouped by coefficient type; (d) top left 4 subimages from (c).



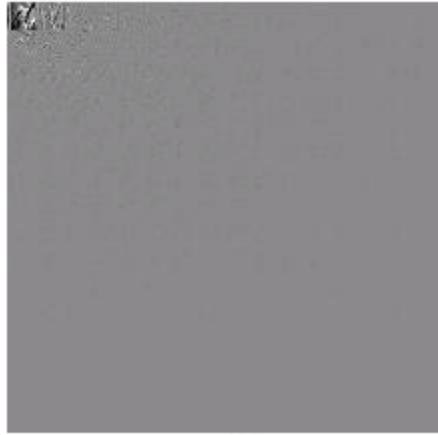
The probabilities p_i and entropies h_i for the 4 subimages from the top left of the 8×8 DCT ((d) of [\[link\]](#)).



(a) Mesh and (b) row plots of the entropies of the subimages of (c) of [\[link\]](#).



(a)



(b)

Lenna transformed by the 4×4 DCT (a) and 16×16 DCT (b).

What is the optimum DCT size?

This is a similar question to: What is the optimum number of levels for the Haar transform?

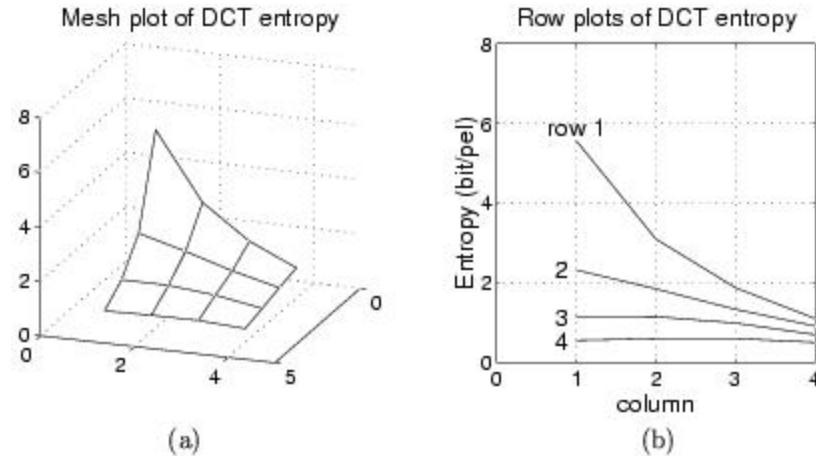
We have analysed Lenna using DCT sizes from 2×2 to 16×16 to investigate this. [\[link\]](#) shows the 4×4 and 16×16 sets of DCT subimages. The 2×2 DCT is identical to the level 1 Haar transform (so see (b) of [\[link\]](#)) and the 8×8 set is in (c) of [\[link\]](#).

[\[link\]](#) and [\[link\]](#) show the mesh plots of the entropies of the subimages in [\[link\]](#).

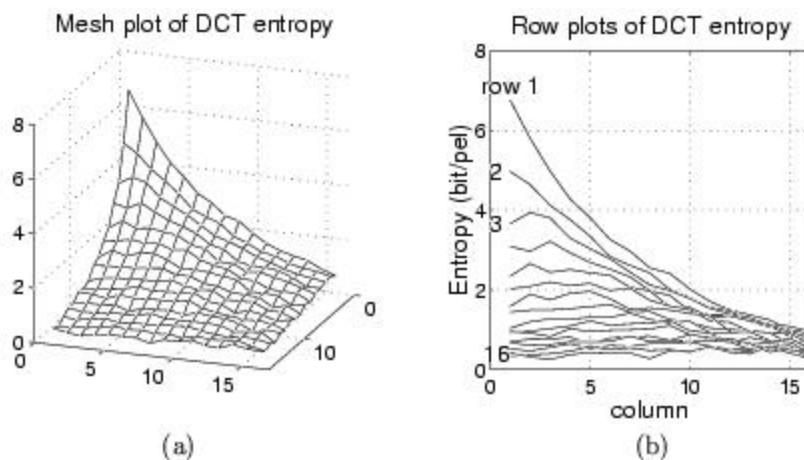
[\[link\]](#) compares the total entropy per pel for the 4 DCT sizes with the equivalent 4 Haar transform sizes. We see that the DCT is significantly better than the rather simpler Haar transform.

As regards the optimum DCT size, from [\[link\]](#), the 16×16 DCT seems to be marginally better than the 8×8 DCT, but subjectively this is not the

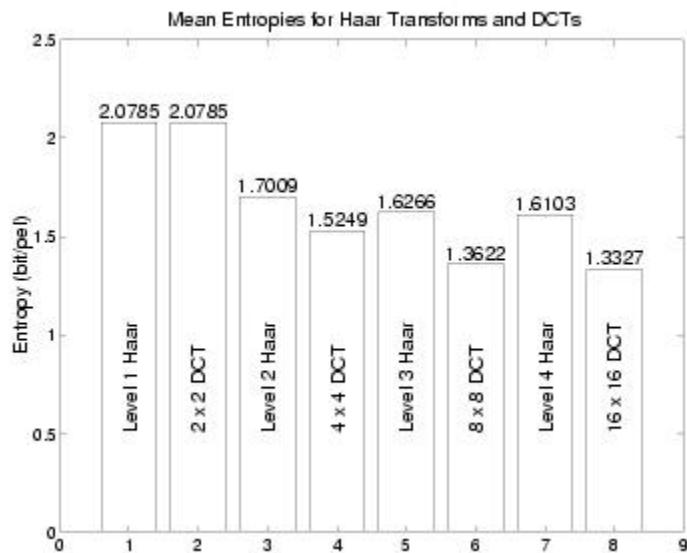
case since quantisation artefacts become more visible as the block size increases. In practise, for a wide range of images and viewing conditions, 8×8 has been found to be the optimum DCT block size and is specified in most current coding standards.



(a) Mesh and (b) row plots of the entropies of the 4×4 DCT in (a) of [\[link\]](#).



(a) Mesh and (b) row plots of the entropies of the 16×16 DCT in (b) of [\[link\]](#).



Comparison of the mean entropies of the Haar transform of Lenna at levels 1 to 4, and of the DCT for sizes from 2×2 to 16×16 pels with $Q_{\text{step}} = 15$

Quantisation of DCT Coefficients

For our discussion of the [2-D DCT](#) we assumed a quantiser step size of 15 to allow direct comparison of entropies with the Haar transform. But what step size do we really need?

[\[link\]](#)(a) and (b) show images reconstructed from the 8×8 DCT of Lenna (see [subfigure \(c\)](#)), when all the DCT coefficients are quantised with step sizes of 15 and 30 respectively. It is difficult to see quantising artefacts in [\[link\]](#)(a) ($Q_{\text{step}} = 15$) but they are quite noticeable in [\[link\]](#)(b) ($Q_{\text{step}} = 30$).

The visibility of the 8×8 DCT basis functions of [subfigure \(a\)](#) in our discussion of the 2-D DCT has been measured (for a 720×576 image viewed from 6 times the image width) and the minimum quantiser steps have been determined which will give artefacts just at the threshold of visibility. The matrices (JPEG Book, p37) for the luminance and chrominance threshold step sizes are:

Equation:

$$Q_{\text{lum}} = \begin{matrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{matrix}$$

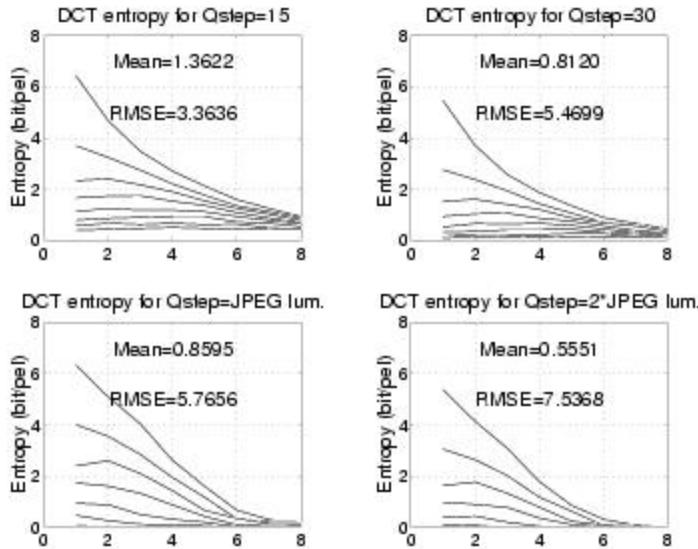
Equation:

$$Q_{\text{chr}} = \begin{matrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{matrix}$$

[link](c) shows the reconstructed image when each of the subimages of (c) is quantised using the corresponding step size from Q_{lum} . It is certainly difficult to detect any quantising artefacts, even though many of the step sizes are greater than $Q_{\text{step}} = 30$, used in [link](b). [link](d) is the reconstructed image using step sizes of $2Q_{\text{lum}}$ and the artefacts are still quite low.

[missing_resource: figure9.png]

Images reconstructed
using the 8×8 DCT with
(a) $Q_{\text{step}} = 15$, (b)
 $Q_{\text{step}} = 30$, (c)
 $Q_{\text{step}} = Q_{\text{lum}}$, the JPEG
luminance matrix, and (d)
 $Q_{\text{step}} = 2Q_{\text{lum}}$.



Plots of the entropies of the 8×8 DCT quantised subimages for the four reconstructed images of [\[link\]](#).

[\[link\]](#) shows the entropies of the 64 quantised subimages used to reconstruct each of the four images in [\[link\]](#). Also given on each plot is the mean entropy (giving the bits/pel for the image) and the rms quantising error between the quantised image and the original.

We see that [\[link\]](#)(c) has about the same mean entropy and rms error as [\[link\]](#)(b), but that its quantising artefacts are much less visible. [\[link\]](#)(d) has similar visibility of artefacts to [\[link\]](#)(b), but has significantly lower entropy and hence **greater compression** (similarly for [\[link\]](#)(c) versus [\[link\]](#)(a)).

This shows the distinct advantages of **subjectively weighted quantisation**, and also that it is unwise to rely too much on the rms error as a measure of image quality.

JPEG Entropy Coding

The entropy plots of the [Quantisation of the DCT coefficients](#) show the theoretical entropies of each DCT sub-band. In practise this would be a poor way to code the data because:

- 64 separate entropy codes would be required (each requiring many extra states to represent run-length coding of zeros).
- The statistics for each code are likely to vary significantly from image to image.
- To transmit the code table for each sub-band as header information would involve a large coding overhead (many extra bits).
- Coding the sub-bands separately does not take account of the correlations which exist between the positions of the non-zero coeffs in one sub-band with those of nearby sub-bands (see [subfigures \(c\) and \(d\)](#) from a previous module).

JPEG uses a clever alternative method of coding, based on combining run-length and amplitude information into a single Huffman code for the whole of the image (except the DC sub-band which is coded separately because its statistics are so different).

The code is applied to each block of 8×8 quantised DCT coeffs from a single 8×8 pel region. The blocks are the coeffs **before reordering** as shown in [subfigure \(b\)](#) of a previous module and comprise one coef from each of the 64 sub-bands.

Each block of 8×8 quantised coeffs is formed into a 1-D vector by zig-zag scanning in the sequence:

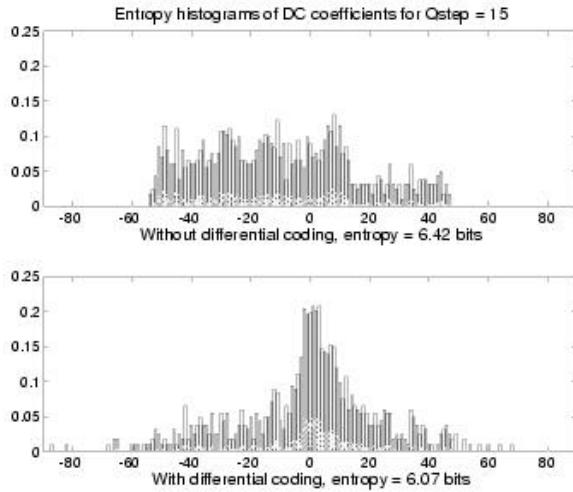
0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

The JPEG Code for DC coeffs

The first coefficient (0) of each block (vector) is the DC coef, which represents the mean value of the pels in the block (see the top left basis function in [subfigure \(a\)](#) from previous discussion).

The DC coefs still exhibit significant local correlations (top left of [subfigure \(b\)](#)), so differential coding is used in which the value to be coded is the **difference** between the current DC coef and that of the previous block - the blocks are scanned from left to right, row by row. The first block in each row is coded with respect to zero.

The histogram of entropies of the DC coef differences is compared in [\[link\]](#) with that of the raw DC coefs from this previous [figure](#). We note the histogram peak around zero and see that the entropy is reduced from 6.42 bits to 6.07 bits.



Histograms of the DC coefficients
from the 8×8 DCT of Lenna,
showing the entropy reduction with
differential coding.

The size of the differences can in theory be up to $\pm(255 \times 8) = \pm(2040)$ if the input pels occupy the range -128 to $+(127)$ (the DCT has a gain of 8 at very low frequencies). Hence the Huffman code table would have to be quite large. JPEG adopts a much smaller code by using a form of floating-point representation, where **Size** is the base-2 exponent and **Additional Bits** are used to code the polarity and precise amplitude as follows:

DC Coef Difference	Size	Typical Huffman codes for Size	Additional Bits (in binary)
0	0	00	-
-1,1	1	010	0,1
-3,-2,2,3	2	011	00,01,10,11
-7,...,-4,4,...,7	3	100	000,...,011,100,...111
-15,...-8,8,...,15	4	101	0000,...,0111,1000,...,1111
:	:	:	:
-1023,...-512,512,...,1023	10	1111 1110	00 0000 0000,...,11 1111 1111
-2047,...-1024,1024,...,2047	11	1 1111 1110	000 0000 0000,...,111 1111 1111

Only Size needs to be Huffman coded in the above scheme, since, within a given Size, all the input values have sufficiently similar probabilities for there to be little gain from entropy coding the Additional Bits (hence they are coded in simple binary as listed). Each coded Size is followed by the appropriate number of Additional Bits (equal to Size) to define the sign and magnitude of the coefficient difference exactly.

There are only 12 Sizes to be Huffman coded, so specifying the code table can be very simple and require relatively few bits in the header.

In JPEG all Huffman code tables are defined in the image header. Each table requires $16 + n$ bytes, where n is the number of codewords in the table.

The first 16 bytes list the number of codewords of each length from 1 to 16 bits (codewords longer than 16 bits are forbidden). The remaining n bytes list the decoded output values of the n codewords in ascending codeword order ($n < 256$).

Hence $16 + 12 = 28$ bytes are needed to specify the code table for DC coefficients.

The JPEG Run-Amplitude Code

The remaining 63 coeffs (the AC coeffs) of each 64-element vector usually contain many zeros and so are coded with a combined run-amplitude Huffman code.

The codeword represents the run-length of zeros before a non-zero coef **and** the Size of that coef. This is then followed by the Additional Bits which define the coef amplitude and sign precisely. Size and Additional Bits are defined just as for DC coeffs.

This 2-dimensional Huffman code (Run, Size) is efficient because there is a strong correlation between the Size of a coef and the expected Run of zeros which precedes it - small coeffs usually follow long runs; larger coeffs tend to follow shorter runs. No single 2-D event is so probable that the Huffman code becomes inefficient.

In order to keep the code table size n below 256, only the following Run and Size values are coded:

Run = 0 → 15

Size = 1 → 10

These require 160 codes. Two extra codes, corresponding to (Run,Size) = (0,0) and (15,0) are used for EOB (End-of-block) and ZRL (Zero run length).

EOB is transmitted after the last non-zero coef in a 64-vector. It is the most efficient way of coding the final run of zeros. It is omitted in the rare case that the final element of the vector is non-zero.

ZRL is transmitted whenever Run > 15, and represents a run of 16 zeros (15 zeros and a zero amplitude coef) which can be part of a longer run of any length. Hence a run of 20 zeros followed by -5 would be coded as

(ZRL) (4,3) 010

When the code tables are defined in the image header, each codeword is assigned to a given (Run,Size) pair by making the decoded output byte **Code Byte** equal to (16 Run + Size).

The default JPEG code for (Run,Size) of AC luminance DCT coefficients is summarised below in order of decreasing code probability:

(Run,Size)	Code Byte (hex)	Code Word (binary)	(Run,Size)	Code Byte (hex)	Code Word (binary)
(0,1)	01	00	(0,6)	06	1111000
(0,2)	02	01	(1,3)	13	1111001
(0,3)	03	100	(5,1)	51	1111010
(EOB)	00	1010	(6,1)	61	1111011
(0,4)	04	1011	(0,7)	07	11111000
(1,1)	11	1100	(2,2)	22	11111001
(0,5)	05	11010	(7,1)	71	11111010
(1,2)	12	11011	(1,4)	14	111110110
(2,1)	21	11100		:	
(3,1)	31	111010	(ZRL)	F0	1111111001
(4,1)	41	111011		:	

As an example, let us code the following 8×8 block:

-13	-3	2	0	0	0	1	0
6	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Concerting this to (DC Size) or (Run,Size) and values for the Additional Bits gives:

```

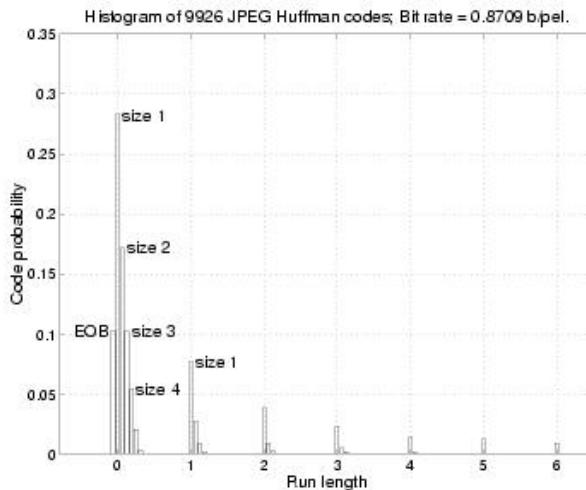
(4) -13 (0,2) -3 (0,3) 6 (2,2) 2 (3,1) -1
(ZRL) (1,1) 1 (EOB)
101 0010 01 00 100 110 11111001 10 111010 0
11111111001 1100 1 1010

```

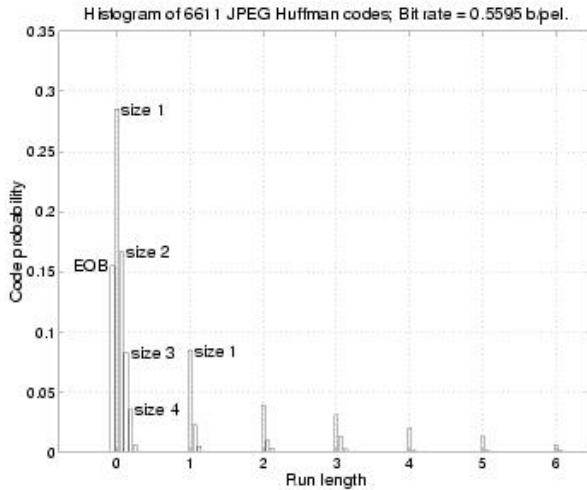
The compressed bitstream for this block is listed on the lower line, assuming that the default Huffman code tables, given above, are used.

[\[link\]](#) shows the histogram of probabilities for the (Run,Size) codewords used to code Lenna using the Q_{lum} quantisation matrix. The bin number represents the decoded byte value.

[\[link\]](#) shows the equivalent histogram when the quantisation matrix is $2Q_{\text{lum}}$.



Histogram of the (Run,Size) codewords for the DCT of Lenna, quantised using Q_{lum} .



Histogram of the (Run,Size) codewords for the DCT of Lenna, quantised using $2Q_{\text{lum}}$.

Note the strong similarity between these histograms, despite the fact that [\[link\]](#) represents only $\frac{2}{3}$ as many events. Only the EOB probability changes significantly, because its probability goes up as the number of events (non-zero coefs) per block goes down.

It turns out that the (Run,Size) histogram remains relatively constant over a wide range of image material and across different regions of each image. This is because of the strong correlation between the run lengths and expected coef sizes. The number of events per block varies considerably depending on the local activity in the image, but the probability distribution of those events (except for EOB) changes much less.

[\[link\]](#) and [\[link\]](#) also give the mean bit rates to code Lenna for the two quantisation matrices. Comparing these with the theoretical entropies from this [figure](#) (lower row) we get:

Q matrix	Mean Entropy (b/pel)	JPEG Bit Rate (b/pel)	JPEG efficiency
Q_{lum}	0.8595	0.8709	98.7%

Q matrix	Mean Entropy (b/pel)	JPEG Bit Rate (b/pel)	JPEG efficiency
$2Q_{\text{lum}}$	0.5551	0.5595	99.21%

Hence we see the high efficiency of the (Run,Size) code at two quite different compression factors. This tends to apply over a wide range of images and compression factors and is an impressive achievement.

There is even very little efficiency lost if a single code table is used for many images, which can avoid the need to transmit the $16 + n$ (168 bytes) of code definition in the header of each image. Using the recommended JPEG default luminance tables (Annex K.3.3) the above efficiencies drop to 97.35% and 95.74% respectively.

Sync and Headers

We have described how individual 8×8 blocks of DCT coefficients are coded. Now we shall briefly look at the sync codes and header information that are needed in order to complete the coding process.

JPEG is rather complex in this aspect, so we shall just give an overview of the basic principles (see the JPEG Book, chapter 7 for the full picture).

JPEG data is divided into **segments**, each of which starts with a 2-byte **marker**.

All markers are byte-aligned - they start on the byte boundaries of the transmission/storage medium. Any variable-length data which precedes a marker is padded with extra ones to achieve this.

The first byte of each marker is `FF` . The second byte defines the type of marker.

To allow for recovery in the presence of errors, it must be possible to detect markers without decoding all of the intervening data. Hence markers must be unique. To achieve this, if an `FF` byte occurs in the middle of a segment, an extra `00` **stuffed** byte is inserted after it and `00` is never used as the second byte of a marker.

Some important markers in the order they are often used are:

Name	Code (hex)	Purpose
SOI	FFD8	Start of image.

Name	Code (hex)	Purpose
COM	FFFE	Comment (segment ignored by decoder). seg , <Text comments>
DQT	FFDB	Define quantisation table(s). seg , < lum , $\text{chr} \dots$ >
SOF_0	FFC0	Start of Baseline DCT frame. seg , <Frame size, no. of components (colours), sub-sampling factors, Q-table selectors>
DHT	FFC4	Define Huffman table(s). seg , <DC Size and AC (Run,Size) tables for each component>
SOS	FFDA	Start of scan. seg , <Huffman table selectors for each component> <Entropy coded DCT blocks>
EOI	FFD9	End of image.

In [\[link\]](#) the data which follows each marker is shown between <> brackets. The first 2-byte word of most segments is the length (in bytes) of the segment, seg . The length of <Entropy coded DCT blocks>, which forms the main bulk of the compressed data, is not specified explicitly, since it may be determined by decoding the entropy codes. This also allows the data to be transmitted with minimal delay, since it is not necessary to determine the total length of the compressed data before any of the DCT block data can be sent.

Long blocks of entropy-coded data are rather prone to being corrupted by transmission errors. To mitigate the worst aspects of this, Restart Markers (FFD0 → FFD7) may be included at regular intervals (say at the start of each row of DCT blocks in the image) so that separate parts of the entropy coded stream may be decoded independently of errors in other parts. The

restart interval, if required, is defined by a DRI (FFDD) marker segment. There are 8 restart markers, which are used in sequence, so that if one (or more) is corrupted by errors, its absence may be easily detected.

The use of multiple scans within each image frame and multiple frames within a given image allows many variations on the ordering and interleaving of the compressed data. For example:

- Chrominance and luminance components may be sent in separate scans or interleaved into a single scan.
- Lower frequency DCT coefs may be sent in one or more scans before higher frequency coefs.
- Coarsely quantised coefs may be sent in one or more scans before finer (refinement) coefs.
- A coarsely sampled frame of the image may be sent initially and then the detail may be progressively improved by adding differentially-coded correction frames of increasing resolution.

The 2-band Filter Bank

Digital filter banks have been actively studied since the 1960s, whereas Wavelet theory is a new subject area that was developed in the 1980s, principally by French and Belgian mathematicians, notably Y. Meyer, I. Daubechies, and S. Mallat. The two topics are now firmly linked and of great importance for signal analysis and compression.

The 2-band Filter Bank

Recall the 1-D Haar transform from our [previous discussion](#).

Equation:

$$\begin{pmatrix} y(1) \\ y(2) \end{pmatrix} = T \begin{pmatrix} x(1) \\ x(2) \end{pmatrix}$$

where $T = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$

We can write this in expanded form as:

Equation:

$$y(1) = \frac{1}{\sqrt{2}} x(1) + \frac{1}{\sqrt{2}} x(2)$$

Equation:

$$y(2) = \frac{1}{\sqrt{2}} x(1) - \frac{1}{\sqrt{2}} x(2)$$

More generally if x is a longer sequence and the results are placed in two separate sequences y_0 and y_1 , we define the process as:

Equation:

$$y_0(n) = \frac{1}{\sqrt{2}}x(n-1) + \frac{1}{\sqrt{2}}x(n)$$

Equation:

$$y_1(n) = \frac{1}{\sqrt{2}}x(n-1) - \frac{1}{\sqrt{2}}x(n)$$

These can be expressed as 2 FIR filters with tap vectors $h_0 = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$ and $h_1 = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{pmatrix}$. Hence as z-transforms, [\[link\]](#) and [\[link\]](#) become:

Equation:

$$Y_0(z) = H_0(z)X(z)$$

where $H_0(z) = \frac{1}{\sqrt{2}} (z^{-1} + 1)$

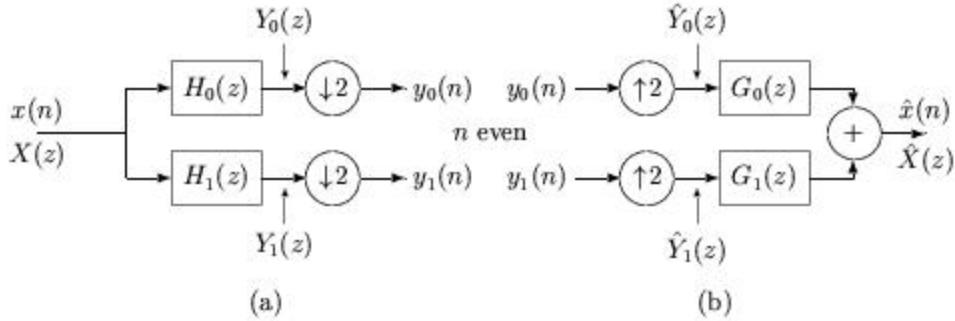
Equation:

$$Y_1(z) = H_1(z)X(z)$$

where $H_1(z) = \frac{1}{\sqrt{2}} (z^{-1} - 1)$. (We shall later extend these filters to be more complicated.)

In practice, we only calculate $y_0(n)$ and $y_1(n)$ at alternate (say even) values of n so that the total number of samples in y_0 and y_1 is the same as in x .

We may thus represent the Haar transform operation by a pair of filters followed by downsampling by 2, as shown in [\[link\]](#)(a). This is known as a 2-band analysis filter bank.



Two-band filter banks for analysis (a) and reconstruction (b).

In [this equation](#) in our discussion of the Haar transform, to reconstruct \mathbf{x} from \mathbf{y} we calculated $\mathbf{x} = T^T \mathbf{y}$. For long sequences this may be written:

Equation:

$$\forall n, n = \text{even} : \left(x(n-1) = \frac{1}{\sqrt{2}} y_0(n) + \frac{1}{\sqrt{2}} y_1(n) \right)$$

Equation:

$$\forall n, n = \text{even} : \left(x(n) = \frac{1}{\sqrt{2}} y_0(n) - \frac{1}{\sqrt{2}} y_1(n) \right)$$

Since $y_0(n)$ and $y_1(n)$ are only calculated at even values of n , we may assume that they are zero at odd values of n . We may then combine [\[link\]](#) and [\[link\]](#) into a single expression for $x(n)$, valid for all n :

Equation:

$$x(n) = \frac{1}{\sqrt{2}} (y_0(n+1) + y_0(n)) + \frac{1}{\sqrt{2}} (y_1(n+1) - y_1(n))$$

or as z-transforms:

Equation:

$$X(z) = G_0(z)Y_0(z) + G_1(z)Y_1(z)$$

where

Equation:

$$\left(G_0(z) = \frac{1}{\sqrt{2}} (z + 1) \right) \wedge \left(G_1(z) = \frac{1}{\sqrt{2}} (z - 1) \right)$$

In [link] the signals $Y_0(z)$ and $Y_1(z)$ are not really the same as $Y_0(z)$ and $Y_1(z)$ in [link] and [link] because those in [link] and [link] have not had alternate samples set to zero. Also, in [link] $X(z)$ is the reconstructed output whereas in [link] and [link] it is the input signal.

To avoid confusion we shall use \widehat{X} , \widehat{Y}_0 and \widehat{Y}_1 for the signals in [link] so it becomes:

Equation:

$$\widehat{X}(z) = G_0(z)\widehat{Y}_0(z) + G_1(z)\widehat{Y}_1(z)$$

We may show this reconstruction operation as upsampling followed by 2 filters, as in [link](b).

If \widehat{Y}_0 and \widehat{Y}_1 are not the same as Y_0 and Y_1 , how do they relate to each other?

Now

Equation:

$$\forall n, n = \text{even} : (\widehat{y}_0(n) = y_0(n))$$

Equation:

$$\forall n, n = \text{odd} : (\widehat{y}_0(n) = 0)$$

Therefore $\widehat{Y}_0(z)$ is a polynomial in z , comprising **only** the terms in even powers of z from $Y_0(z)$. This may be written as:

Equation:

$$\begin{aligned}\widehat{Y}_0(z) &= \sum_{\text{even } n} y_0(n)z^{-n} \\ &= \sum_{\text{all } n} \frac{1}{2} (y_0(n)z^{-n} + y_0(n)(-z)^{-n}) \\ &= \frac{1}{2} (Y_0(z) + Y_0(-z))\end{aligned}$$

Similarly

Equation:

$$\widehat{Y}_1(z) = \frac{1}{2} (Y_1(z) + Y_1(-z))$$

This is our general model for downsampling by 2, followed by upsampling by 2 as defined in [\[link\]](#) and [\[link\]](#).

Perfect Reconstruction (PR)

We are now able to generalize our analysis for arbitrary filters H_0 , H_1 , G_0 and G_1 . Substituting [this equation](#) and [this equation](#) in our discussion of 2-band filter bank into [this earlier equation](#) and then using [this equation](#) and [this equation](#) from the same discussion, we get:

Equation:

$$\begin{aligned}\widehat{X}(z) &= \frac{1}{2}G_0(z)(Y_0(z) + Y_0(-z)) + \frac{1}{2}G_1(z)(Y_1(z) + Y_1(-z)) \\ &= \frac{1}{2}G_0(z)H_0(z)X(z) + \frac{1}{2}G_0(z)H_0(-z)X(-z) + \frac{1}{2}G_1(z)H_1(z)X(z) + \frac{1}{2}G_1(z)H_1(-z)X(-z) \\ &= \frac{1}{2}X(z)(G_0(z)H_0(z) + G_1(z)H_1(z)) + \frac{1}{2}X(-z)(G_0(z)H_0(-z) + G_1(z)H_1(-z))\end{aligned}$$

If we require $\widehat{X}(z) \equiv X(z)$ - the Perfect Reconstruction (PR) condition - then:

Equation:

$$G_0(z)H_0(z) + G_1(z)H_1(z) \equiv 2$$

and

Equation:

$$G_0(z)H_0(-z) + G_1(z)H_1(-z) \equiv 0$$

Identity [\[link\]](#) is known as the **anti-aliasing condition** because the term in $X(-z)$ in [\[link\]](#) is the unwanted aliasing term caused by down-sampling y_0 and y_1 by 2.

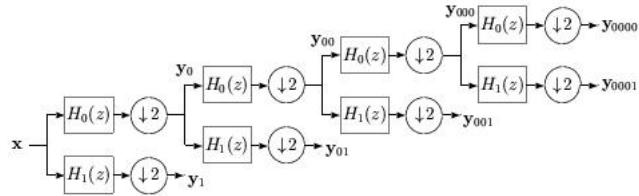
It is straightforward to show that the expression for H_0 , H_1 , G_0 and G_1 , given in [this equation](#), [this equation](#), [this equation](#) and [this equation](#) for the filters based on the Haar transform, satisfy [\[link\]](#) and [\[link\]](#). They are the simplest set of filters which do.

Before we look at more complicated PR filters, we examine how the filter structures of [this figure](#) may be extended to form a binary filter tree (and the discrete wavelet transform).

The Binary Filter Tree

Recall that for image compression (see [The 2-band Filter Bank](#)), the purpose of the 2-band filter bank in the Haar transform is to compress most of the signal energy into the low-frequency band.

We may achieve greater compression if the low band is further split into two. This may be repeated a number of times to give the binary filter tree, shown with 4 levels in [\[link\]](#).



Extension of the 2-band filter bank into a binary filter tree.

In 1-D, this is analogous to the way the [2-D Haar transform](#) was extended to the [multi-level Haar transform](#).

For an N -sample input vector \mathbf{x} , the sizes and bandwidths of the signals of the 4-level filter tree are:

Signal	No. of samples	Approximate pass band
x	N	$0 \rightarrow \frac{1}{2} f_s$
y_1	$\frac{N}{2}$	$\frac{1}{4} \rightarrow \frac{1}{2} f_s$
y_{01}	$\frac{N}{4}$	$\frac{1}{8} \rightarrow \frac{1}{4} f_s$
y_{001}	$\frac{N}{8}$	$\frac{1}{16} \rightarrow \frac{1}{8} f_s$
y_{0001}	$\frac{N}{16}$	$\frac{1}{32} \rightarrow \frac{1}{16} f_s$
y_{0000}	$\frac{N}{16}$	$0 \rightarrow \frac{1}{32} f_s$

Because of the downsampling (decimation) by 2 at each level, the total number of output samples = N , regardless of the number of levels in the tree.

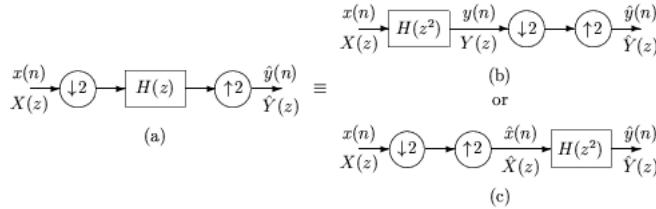
The H_0 filter is normally designed to be a lowpass filter with a passband from 0 to approximately $\frac{1}{4}$ of the input sampling frequency for that stage; and H_1 is a highpass (bandpass) filter with a pass band approximately from $\frac{1}{4}$ to $\frac{1}{2}$ of the input sampling frequency.

When formed into a 4-level tree, the filter outputs have the approximate pass bands given in [\[link\]](#). The final output y_{0000} is a lowpass signal, while the other outputs are all bandpass signals, each covering a band of approximately one octave.

An inverse tree, mirroring [link], may be constructed using filters G_0 and G_1 instead of H_0 and H_1 , as shown for just one level in part (b) of [this figure](#). If the PR conditions of [this previous equation](#) and [this previous equation](#) are satisfied, then the output of each level will be identical to the input of the equivalent level in [link], and the final output will be a perfect reconstruction of the input signal.

Multi-rate filtering theorem

To calculate the impulse and frequency responses for a multistage network with downsampling at each stage, as in [link], we must first derive an important theorem for multirate filters.



Multi-rate filtering - the result of shifting a filter ahead of a downsampling operation or after an upsampling operation.

The downsample-filter-upsample operation of [link](a) is equivalent to either the filter-downsample-upsample operation of [link](b) or the downsample-upsample-filter operation of [link](c), if the filter is changed from $H(z)$ to $H(z^2)$.

From [link](a):

Equation:

$$\hat{y}(n) = \begin{cases} \sum_i x(n-2i)h(i) & \text{if } n \text{ is even} \\ 0 & \text{if } n \text{ is odd} \end{cases}$$

Take z-transforms:

Equation:

$$\begin{aligned} \hat{Y}(z) &= \sum_n \hat{y}(n)z^{-n} \\ &= \sum_{\text{even } n} \sum_i x(n-2i)h(i)z^{-n} \end{aligned}$$

Reverse the order of summation and let $m = n - 2i$: therefore,

Equation:

$$\begin{aligned} \hat{Y}(z) &= \sum_i h(i) \sum_{\text{even } m} x(m)z^{-m}z^{-2i} \\ &= \sum_i h(i)z^{-2i} \sum_{\text{even } m} x(m)z^{-m} \\ &= H(z^2) \frac{1}{2} (X(z) + X(-z)) \\ &= \frac{1}{2} \left(H(z^2)X(z) + H((-z)^2)X(-z) \right) \\ &= \frac{1}{2} (Y(z) + Y(-z)) \end{aligned}$$

where $Y(z) = H(z^2)X(z)$

This describes the operations of [link](b). Hence the first result is proved.

The result from line 3 in [link]

Equation:

$$\begin{aligned}\widehat{Y}(z) &= \frac{1}{2} (X(z) + X(-z))H(z^2) \\ &= \widehat{X}(z)H(z^2)\end{aligned}$$

shows that the filter $H(z^2)$ may be placed after the down/up-sampler as in [link](c), which proves the second result.

General results for M:1 subsampling

It can be shown that:

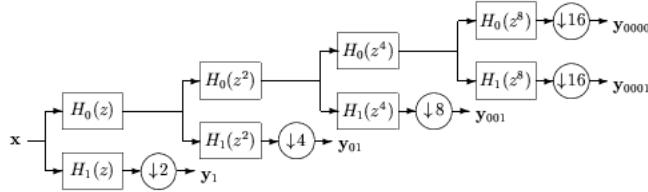
- $H(z)$ becomes $H(z^M)$ if shifted ahead of an M:1 downsample or following an M:1 upsample.
- M:1 down/up-sampling of a signal $X(z)$ produces:

Equation:

$$\widehat{X}(z) = \frac{1}{M} \sum_{m=0}^{M-1} X\left(ze^{\frac{i2\pi m}{M}}\right)$$

Transformation of the filter tree

Using the result of [link], [link] can be redrawn as in [link] with all downsamplers moved to the outputs. (Note [link] requires much more computation than [link].)



Binary filter tree, transformed so that all downsampling operations occur at the outputs.

We can now calculate the transfer function to each output (before the downsamplers) as:

Equation:

$$H_{01}(z) = H_0(z)H_1(z^2)$$

Equation:

$$H_{001}(z) = H_0(z)H_0(z^2)H_1(z^4)$$

Equation:

$$H_{0001}(z) = H_0(z)H_0(z^2)H_0(z^4)H_1(z^8)$$

Equation:

$$H_{0000}(z) = H_0(z)H_0(z^2)H_0(z^4)H_0(z^8)$$

In general the transfer functions to the two outputs at level k of the tree are given by:

Equation:

$$H_{k,1} = \prod_{i=0}^{k-2} H_0(z^{2^i})H_1(z^{2^{k-1}})$$

Equation:

$$H_{k,0} = \prod_{i=0}^{k-1} H_0(z^{2^i})$$

For the Haar filters of [this equation](#) and [this equation](#) from our discussion of the 2-band filter bank, the transfer functions to the outputs of the 4-level tree become:

Equation:

$$H_{01}(z) = \frac{1}{2} (z^{-3} + z^{-2} - z^{-1} + 1)$$

Equation:

$$H_{001}(z) = \frac{1}{2 \times \sqrt{2}} (z^{-7} + z^{-6} + z^{-5} + z^{-4} - z^{-3} + z^{-2} + z^{-1} + 1)$$

Equation:

$$H_{0001}(z) = \frac{1}{4} (z^{-15} + z^{-14} + z^{-13} + z^{-12} + z^{-11} + z^{-10} + z^{-9} + z^{-8} - z^{-7} + z^{-6} + z^{-5} + z^{-4} + z^{-3} + z^{-2})$$

Equation:

$$H_{0000}(z) = \frac{1}{4} (z^{-15} + z^{-14} + z^{-13} + z^{-12} + z^{-11} + z^{-10} + z^{-9} + z^{-8} + z^{-7} + z^{-6} + z^{-5} + z^{-4} + z^{-3} + z^{-2})$$

Wavelets

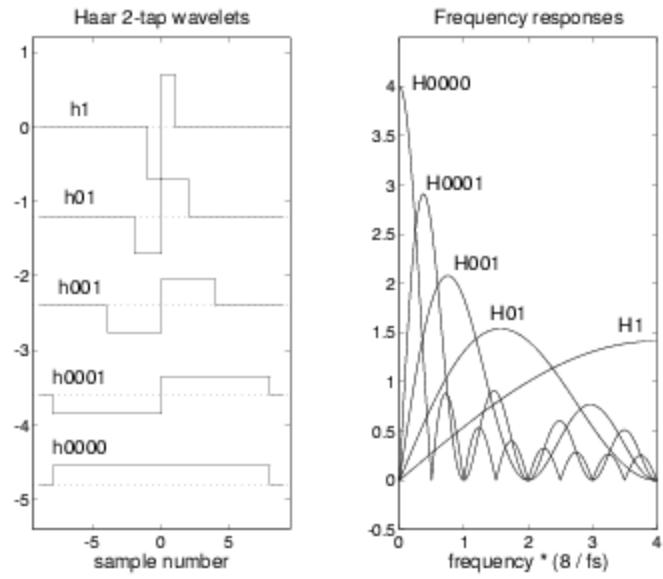
The process of creating the outputs y_1 to y_{0000} from \mathbf{x} is known as the **discrete wavelet transform (DWT)**; and the reconstruction process is the **inverse DWT**.

The word **wavelet** refers to the impulse response of the cascade of filters which leads to a given bandpass output. The frequency response of the wavelet at level k is obtained by substituting $z = e^{i\omega T_s}$ in the z-transfer function $H_{k,1}$ from [this equation](#) and [this equation](#) from our discussion of the binary filter tree. T_s is the sampling period at the input to the filter tree.

Since the frequency responses of the bandpass bands are scaled down by 2:1 at each level, their impulse responses become longer by the same factor at each level, **BUT** their shapes remain very similar. The basic impulse response wave shape is almost independent of scale and known as the **mother wavelet**.

The impulse response to a lowpass output $H_{k,0}$ is called the **scaling function** at level k .

[\[link\]](#) shows these effects using the impulse responses and frequency responses for the five outputs of the 4-level tree of Haar filters, based on the z-transforms given in this group of [equations](#). Notice the abrupt transitions in the middle and at the ends of the Haar wavelets. These result in noticeable **blocking** artefacts in decompressed images (as in part (b) of this previous [figure](#)).



Impulse responses and frequency responses of the 4-level tree of Haar filters.

Good Filters / Wavelets

Our main aim now is to search for better filters / wavelets which result in compression performance that rivals or beats the DCT.

We assume that perfect reconstruction is a prime requirement, so that the only image degradations are caused by coefficient quantisation, and may be made as small as we wish by increasing bit rate.

We start our search with the two PR identities from our discussion of [Perfect Reconstruction](#), which we repeat here:

Equation:

$$G_0(z)H_0(z) + G_1(z)H_1(z) \equiv 2$$

and

Equation:

$$G_0(z)H_0(-z) + G_1(z)H_1(-z) \equiv 0$$

The usual way of satisfying the anti-aliasing condition ([\[link\]](#)), while permitting H_0 and G_0 to have lowpass responses (passband where $\Re(z) > 0$) and H_1 and G_1 to have highpass responses (passband where $\Re(z) < 0$), is with the following relations:

Equation:

$$H_1(z) = z^{-k}G_0(-z)$$

and

Equation:

$$G_1(z) = z^kH_0(-z)$$

where k must be odd so that:

$$G_0(z)H_0(-z) + G_1(z)H_1(-z) = G_0(z)H_0(-z) + z^kH_0(-z)(-z)^{-k}G_0(z) = 0$$

Now define the lowpass product filter:

Equation:

$$P(z) = H_0(z)G_0(z)$$

and substitute relations [\[link\]](#) and [\[link\]](#) into identity [\[link\]](#) to get:

Equation:

$$\begin{aligned}
G_0(z)H_0(z) + G_1(z)H_1(z) &= G_0(z)H_0(z) + H_0(-z)G_0(-z) \\
&= P(z) + P(-z) \\
&= 2
\end{aligned}$$

This requires all $P(z)$ terms in even powers of z to be zero, except the z^0 term which should be 1. The $P(z)$ terms in odd powers of z may take any desired values since they cancel out in [\[link\]](#).

A further constraint on $P(z)$ is that it should be zero phase, in order to minimise the visibility of any distortions due to the high-band being quantised to zero. Hence $P(z)$ should be of the form:

Equation:

$$P(z) = \dots + p_5z^5 + p_3z^3 + p_1z + 1 + p_1z^{-1} + p_3z^{-3} + p_5z^{-5} + \dots$$

The design of a set of PR filters H_0 , H_1 and G_0 , G_1 can now be summarised as:

1. Choose a set of coefficients $p_1, p_3, p_5 \dots$ to give a zero-phase lowpass product filter $P(z)$ with desirable characteristics. (This is non-trivial and is discussed below.)
2. Factorize $P(z)$ into $H_0(z)$ and $G_0(z)$, preferably so that the two filters have similar lowpass frequency responses.
3. Calculate $H_1(z)$ and $G_1(z)$ from [\[link\]](#) and [\[link\]](#).

It can help to simplify the tasks of choosing $P(z)$ and factorising it if, based on the zero-phase requirement, we transform $P(z)$ into $P_t(Z)$ such that:

Equation:

$$\begin{aligned}
P(z) &= P_t(Z) \\
&= 1 + p_{t,1}Z + p_{t,3}Z^3 + p_{t,5}Z^5 + \dots
\end{aligned}$$

where $Z = \frac{1}{2}(z + z^{-1})$. To calculate the frequency response of P_t , let $z = e^{i\omega T_s}$: therefore,

Equation:

$$\begin{aligned}
Z &= \frac{1}{2}(e^{i\omega T_s} + e^{-(i\omega T_s)}) \\
&= \cos(\omega T_s)
\end{aligned}$$

This is a purely real function of ω , varying from 1 at $\omega = 0$ to -1 at $\omega T_s = \pi$ (half the sampling frequency).

A Belgian mathematician, Ingrid Daubechies, did much pioneering work on wavelets in the 1980s. She discovered that to achieve smooth wavelets after many levels of the binary tree, the lowpass filters $H_0(z)$ and $G_0(z)$ must both have a number of zeros at half the sampling frequency (at $z = -1$). These will also be zeros of $P(z)$, and so $P_t(z)$ will have zeros at $Z = -1$.

The simplest case is a single zero at $Z = -1$, so that $P_t(z) = 1 + Z$. Therefore,

$$P(z) = \frac{1}{2} (z + 2 + z^{-1}) = \frac{1}{2} (z + 1) (1 + z^{-1}) = G_0(z)H_0(z)$$

which gives the familiar Haar filters.

As we have seen, the Haar wavelets have significant discontinuities so we need to add more zeros at $Z = -1$. However to maintain PR, we must also ensure that all terms in even powers of Z are zero, so the next more complicated P_t must be of the form:

Equation:

$$\begin{aligned} P_t(z) &= (1 + Z)^2 (1 + \alpha Z) \\ &= 1 + (2 + \alpha)Z + (1 + 2\alpha)Z^2 + \alpha Z^3 \end{aligned}$$

if $\alpha = -\frac{1}{2}$ to suppress the term in Z^2 ,

$$P_t(z) = 1 + \frac{3}{2}Z - \frac{1}{2}Z^3$$

If we allocate the factors of P_t such that $(1 + Z)$ gives H_0 and $(1 + Z)(1 + \alpha Z)$ gives G_0 , we get:

Equation:

$$H_0(z) = \frac{1}{2} (z + 2 + z^{-1})$$

Equation:

$$G_0(z) = \frac{1}{8} (z + 2 + z^{-1}) (-z + 4 - z^{-1}) \left(\frac{1}{8} (-z^2 + 2z + 6 + 2z^{-1} - z^{-2}) \right)$$

Using [\[link\]](#) and [\[link\]](#) with $k = 1$, the corresponding highpass filters then become:

Equation:

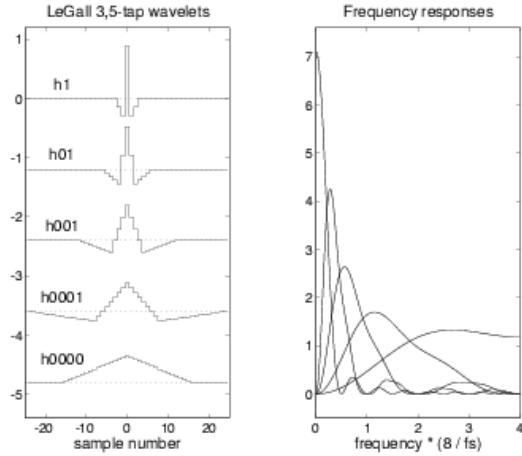
$$\begin{aligned} G_1(z) &= zH_0(-z) \\ &= \frac{1}{2}z(-z + 2 - z^{-1}) \end{aligned}$$

Equation:

$$\begin{aligned} H_1(z) &= z^{-1}G_0(-z) \\ &= \frac{1}{8}z^{-1}((-z^2) - 2z + 6 - 2z^{-1} - z^{-2}) \end{aligned}$$

This is often known as the **LeGall 3,5-tap filter set**, since it was first published in the context of 2-band filter banks by Didier LeGall in 1988.

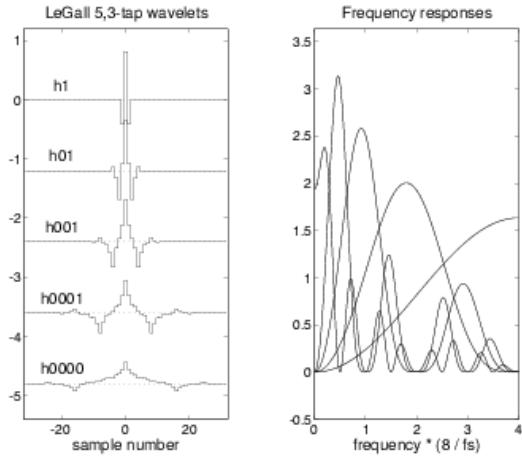
The wavelets of the LeGall 3,5-tap filters, H_0 and H_1 above, and their frequency responses are shown in [\[link\]](#). The scaling function (bottom left) converges to a pure triangular pulse and the wavelets are the superposition of two triangular pulses.



Impulse responses and frequency responses of the 4-level tree of LeGall 3,5-tap filters.

The triangular scaling function produces linear interpolation between consecutive lowband coefficients and also causes the wavelets to be linear interpolations of the coefficients of the H_1 filter, -1, -2, 6, -2, -1 (scaled appropriately).

These wavelets have quite desirable properties for image compression (note the absence of waveform discontinuities and the much lower sidelobes of the frequency responses), and they represent probably the simplest useful wavelet design. Unfortunately there is one drawback -- the inverse wavelets are not very good. These are formed from the LeGall 5,3-tap filter pair, G_0 and G_1 above, whose wavelets and frequency responses are shown in [\[link\]](#).



Impulse responses and frequency responses of the 4-level tree of

LeGall 5,3-tap filters.

The main problem is that the wavelets do not converge after many levels to a smooth function and hence the frequency responses have large unwanted sidelobes. The jaggedness of the scaling function and wavelets causes highly visible coding artefacts if these filters are used for reconstruction of a compressed image.

However the allocation of the factors of $P_t(Z)$ to H_0 and G_0 is a free design choice, so we may swap the factors (and hence swap G and H) in order that the smoother 3,5-tap filters become G_0 , G_1 and are used for reconstruction. We shall show later that this leads to a good low-complexity solution for image compression and that the jaggedness of the analysis filters is not critical.

Unbalance between analysis and reconstruction filters / wavelets is nevertheless often regarded as being undesirable, particularly as it prevents the filtering process from being represented as an orthonormal transformation of the input signal (since an orthonormally transformed signal may be reconstructed simply by transposing the transform matrix). An unbalanced PR filter system is often termed a **bi-orthogonal transformation**.

We now consider ways to reduce this unbalance.

Filters with balanced H and G frequency responses (but non-linear phase responses) - Daubechies wavelets:

In the above analysis, we used the factorisation of $P_t(Z)$ to give us H_0 and G_0 . This always gives unbalanced factors if terms of P_t in even powers of Z are zero.

However each of these factors in Z may itself be factorised into a pair of factors in z , since:
Equation:

$$\begin{aligned} (\alpha z + 1) (1 + \alpha z^{-1}) &= \alpha z + 1 + \alpha^2 + \alpha z^{-1} \\ &= 1 + \alpha^2 + 2\alpha Z \\ &= (1 + \alpha^2) (1 + \beta Z) \end{aligned}$$

where $\beta = \frac{2\alpha}{1+\alpha^2}$.

For each factor of $P_t(Z)$, we may allocate one of its z subfactors to $H_0(z)$ and the other to $G_0(z)$. Where roots of $P_t(Z)$ are complex, the subfactors must be allocated in conjugate pairs so that H_0 and G_0 remain purely real.

Since the subfactors occur in reciprocal pairs (roots at $z = \alpha$ and α^{-1}), we find that
Equation:

$$G_0(z) = H_0(z^{-1})$$

which means that the impulse response of G_0 is the time-reverse of H_0 .

Therefore the frequency responses are related by $G_0(e^{i\omega T_s}) = H_0(e^{-(i\omega T_s)})$.

Hence the magnitudes of the frequency responses are the same, and their phases are opposite. It may be shown that this is sufficient to obtain **orthogonal** wavelets, but unfortunately the separate filters are no longer zero (or linear) phase. (Linear phase is zero phase with an arbitrary delay z^{-k} .)

Daubechies wavelets may be generated in this way, with the added constraint that the maximum number of zeros of $P_t(Z)$ are placed at $Z = -1$ (producing pairs of zeros of $P(z)$ at $z = -1$), consistent with terms in even powers of Z being zero.

If $P_t(Z)$ is of order $2K - 1$, then it may have K zeros at $Z = -1$ such that

Equation:

$$P_t(Z) = (1 + Z)^K R_t(Z)$$

where $R_t(Z)$ is of order $K - 1$ and its $K - 1$ roots may be chosen such that terms of $P_t(Z)$ in the $K - 1$ even powers of Z are zero.

[link] is the $K = 2$ solution to [link]. Therefore, $R_t(Z) = 1 - \frac{1}{2}Z$ so $\beta = -\frac{1}{2}$ and, from [link], the factors of $R(z)$ are

$$R(z) = \frac{(\alpha z + 1)(1 + \alpha z^{-1})}{1 + \alpha^2}$$

where $\alpha = \sqrt{3} - 2$. Also

$$(1 + Z)^2 = \frac{1}{2}(Z + 1)^2 \frac{1}{2}(1 + z^{-1})^2$$

Hence

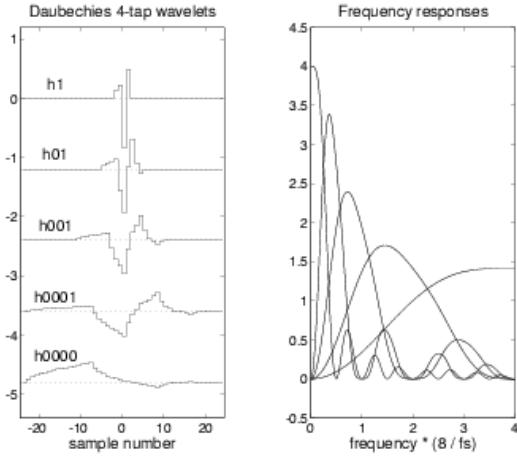
$$H_0(z) = \frac{1}{2\sqrt{1 + \alpha^2}} (1 + z^{-1})^2 (1 + \alpha z^{-1}) = 0.4830 + 0.8365z^{-1} + 0.2241z^{-2} - 0.1294z^{-3}$$

and

$$H_1(z) = z^{-3}G_0(-z) = z^{-3}H_0(-z^{-1}) = 0.1294 + 0.2241z^{-1} - 0.8365z^{-2} + 0.4830z^{-3}$$

The wavelets and frequency responses for these filters are shown in [link]. It is clear that the wavelets and scaling function are no longer linear phase and are less smooth than those for the

LeGall 3,5-tap filters. The frequency responses also show worse sidelobes. The G_0, G_1 filters give the time reverse of these wavelets and identical frequency responses.



Impulse responses and frequency responses of the 4-level tree of Daubechies 4-tap filters.

Higher order Daubechies filters achieve smooth wavelets but they still suffer from non-linear phase. This tends to result in more visible coding artefacts than linear phase filters, which distribute any artefacts equally on either side of sharp edges in the image.

Linear phase filters also allow an elegant technique, known as symmetric extension, to be used at the outer edges of images, where wavelet filters would otherwise require the size of the transformed image to be increased to allow for convolution with the filters. Symmetric extension assumes that the image is reflected by mirrors at each edge, so that an infinitely tessellated plane of reflected images is generated. Reflections avoid unwanted edge discontinuities. If the filters are linear phase, then the DWT coefficients also form reflections and no increase in size of the transformed image is necessary to accomodate convolution effects.

Filters with linear phase and nearly balanced frequency responses:

To ensure that the filters H_0, H_1 and G_0, G_1 are linear phase, the factors in Z must be allocated to H_0 or G_0 as a whole and not be split, as was done for the Daubechies filters. In this way the symmetry between z and z^{-1} is preserved in all filters.

Perfect balance of frequency responses between H_0 and G_0 is then not possible, if PR is preserved, but we have found a factorisation of $P_t(Z)$ which achieves near balance of the responses.

This is:

Equation:

$$P_t(Z) = (1 + Z) (1 + aZ + bZ^2) (1 + Z) (1 + cZ)$$

This is a 5th order polynomial, and if the terms in Z^2 and Z^4 are to be zero, there are two constraints on the 3 unknowns $[a, b, c]$ so that one of them (say c) may be regarded as a free parameter. These constraints require that:

Equation:

$$a = -\frac{(1 + 2c)^2}{2(1 + c)^2}$$

and

Equation:

$$b = \frac{c(1 + 2c)}{2(1 + c)^2}$$

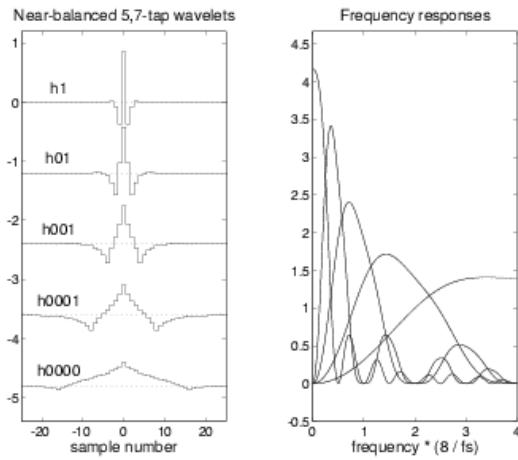
c may then be adjusted to give maximum similarity between the left and right pairs of factors in [\[link\]](#) as Z varies from 1 to -1 (ωT_s varies from 0 to π).

It turns out that $c = -\frac{2}{7}$ gives good similarity and when substituted into [\[link\]](#), [\[link\]](#) and [\[link\]](#) gives:

Equation:

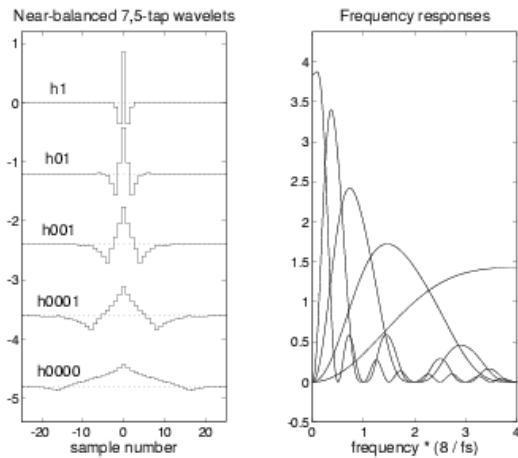
$$P_t(Z) = \frac{1}{50} (50 + 41Z - 15Z^2 - 6Z^3) \frac{1}{7} (7 + 5Z - 2Z^2)$$

We get $G_0(z)$ and $H_0(z)$ by substituting $Z = \frac{1}{2} (z + z^{-1})$ into these two polynomial factors. This results in 5,7-tap filters whose wavelets and frequency responses are shown in [\[link\]](#).



Impulse responses and frequency responses of the 4-level tree of near-balanced 5,7-tap filters.

The near balance of the responses may be seen from [\[link\]](#) which shows the alternative 7,5-tap versions (i.e. with H and G swapped). It is quite difficult to spot the minor differences between these figures.



Impulse responses and frequency responses of the 4-level tree of near-balanced 7,5-tap filters.

Smoother Wavelets

In all of the above designs we have used the substitution $Z = \frac{1}{2} (z + z^{-1})$. However other substitutions may be used to create improved wavelets. To preserve PR, the substitution should contain only odd powers of z (so that odd powers of Z should produce only odd powers of z), and to produce zero phase, the coefficients of the substitution should be symmetric about z^0 .

A substitution, which can give much greater flatness near $z = \pm(1)$ while still satisfying $Z = \pm(1)$ when $z = \pm(1)$, is:

Equation:

$$Z = pz^3 - \frac{1}{2} (z + z^{-1}) + pz^{-3}$$

Z then becomes the following function of frequency when $z = e^{i\omega T_s}$:

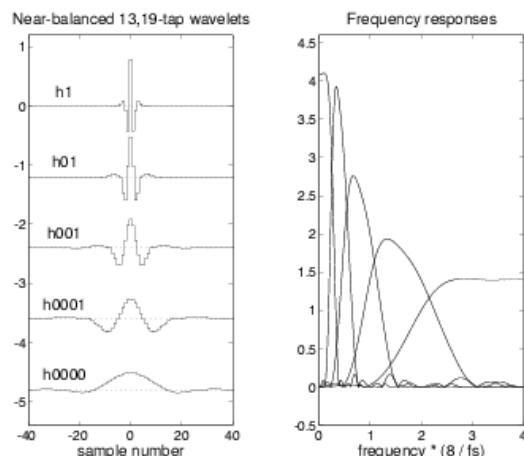
Equation:

$$Z = 1 \cos(\omega T_s) + 2p \cos(3\omega T_s)$$

A high degree of flatness (with some ripple) is achieved near $\omega T_s = 0$ and π , when $p = -\frac{3}{32}$. This is equivalent to more zeros near $z = -1$ for each $(Z + 1)$ factor than when $Z = \frac{1}{2} (z + z^{-1})$ is used.

The 2nd order factor in $P_t(Z)$ now produces terms from z^6 to z^{-6} and the 3rd order factor produces terms from z^9 to z^{-9} . Hence the filters become 13 and 19 tap filters, although 2 taps of each are zero and the outer two taps of the 19-tap filter are very small ($\approx (10^{-4})$).

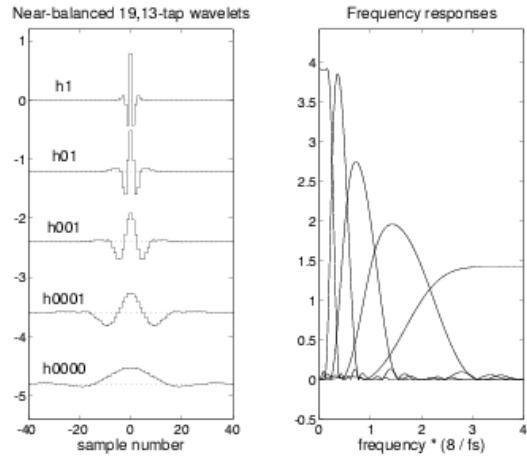
[\[link\]](#) shows the wavelets and frequency responses of the 13,19-tap filters, obtained by substituting [\[link\]](#) into [\[link\]](#). Note the smoother wavelets and scaling function and the much lower sidelobes in the frequency responses from these higher order filters.



Impulse responses and frequency

responses of the 4-level tree of near-balanced 13,19-tap filters.

[\[link\]](#) demonstrates that the near balanced properties of [\[link\]](#) are preserved in the high order filters.



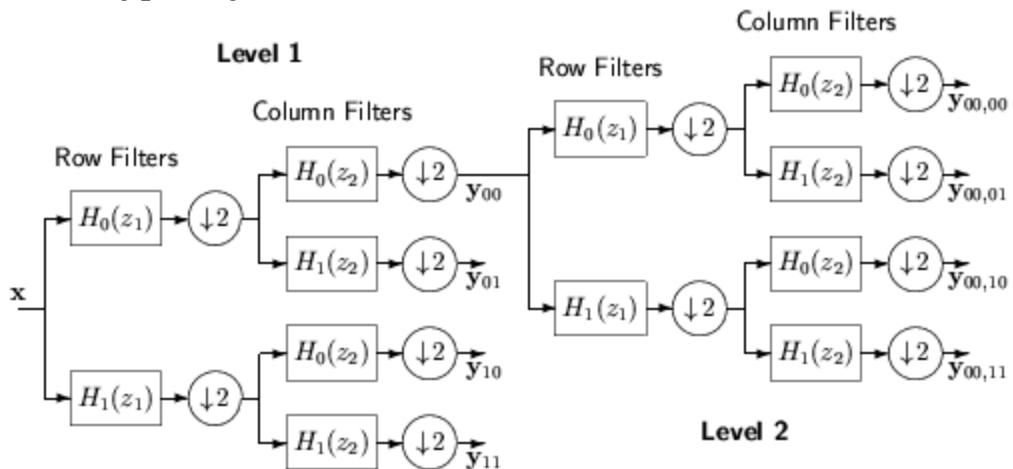
Impulse responses and frequency responses of the 4-level tree of near-balanced 19,13-tap filters.

There are many other types of wavelets with varying features and complexities, but we have found the examples given to be near optimum for image compression.

The 2-D DWT

We have already seen in our discussion of [The Haar Transform](#) how the 1-D Haar transform (or wavelet) could be extended to 2-D by filtering the rows and columns of an image separably.

All 1-D 2-band wavelet filter banks can be extended in a similar way. [\[link\]](#) shows two levels of a 2-D filter tree. The input image at each level is split into 4 bands (Lo-Lo = y_{00} , Lo-Hi = y_{01} , Hi-Lo = y_{10} , and Hi-Hi = y_{11}) using the lowpass and highpass wavelet filters on the rows and columns in turn. The Lo-Lo band subimage y_{00} is then used as the input image to the next level. Typically 4 levels are used, as for the Haar transform.



Two levels of a 2-D filter tree, formed from 1-D lowpass (H_0) and highpass (H_1) filters.

Filtering of the rows of an image by $H_a(z_1)$ and of the columns by $H_b(z_2)$, where $a, b = 0$ or 1 , is equivalent to filtering by the 2-D filter:

Equation:

$$H_{ab}(z_1, z_2) = H_a(z_1)H_b(z_2)$$

In the spatial domain, this is equivalent to convolving the image matrix with the 2-D impulse response matrix

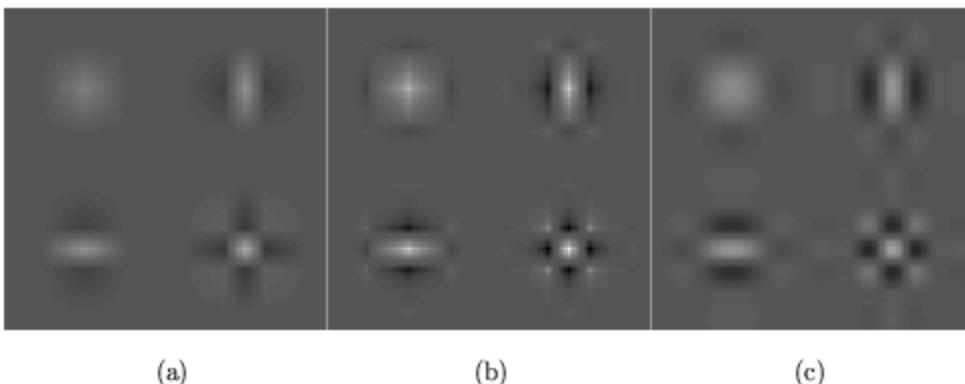
Equation:

$$h_{a,b} = \mathbf{h}_a \mathbf{h}_b^T$$

where \mathbf{h}_a and \mathbf{h}_b are column vectors of the 1-D filter impulse responses. However note that performing the filtering separably (i.e. as separate 1-D filterings of the rows and columns) is much more computationally efficient.

To obtain the impulse responses of the four 2-D filters at each level of the 2-D DWT we form $h_{a,b}$ from \mathbf{h}_0 and \mathbf{h}_1 using [\[link\]](#) with $ab = 00, 01, 10$ and 11 .

LeGall 3,5-tap, and near balanced 5,7-tap and 13,19-tap 2-D wavelets at level 4



2-D impulse responses of the level-4 wavelets and scaling functions derived from the LeGall 3,5-tap filters (a), and the near-balanced 5,7-tap (b) and 13,19-tap (c) filters.

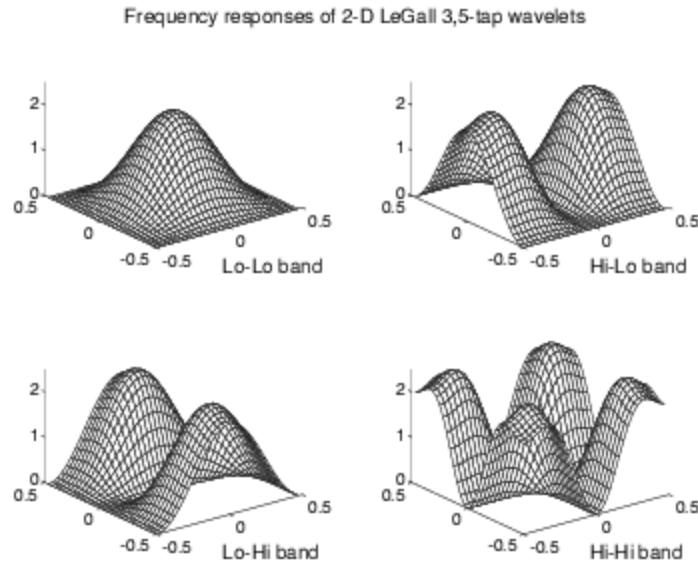
[\[link\]](#) shows the impulse responses at level 4 as images for three 2-D wavelet filter sets, formed from the following 1-D wavelet filter sets:

1. The LeGall 3,5-tap filters: H_0 and H_1 from [these equations](#), and [these equations](#) in our discussion of Good Filters / Wavelets.
2. The near-balanced 5,7-tap filters: substituting $Z = \frac{1}{2} (z + z^{-1})$ into [this previous equation](#).

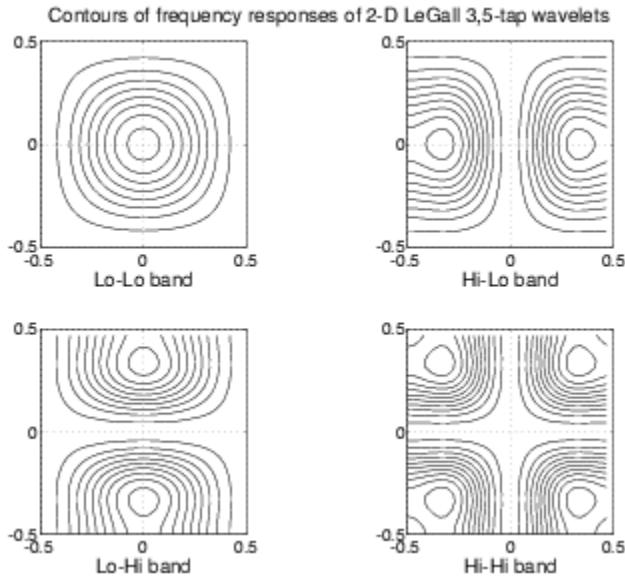
3. The near-balanced 13,19-tap filters: substituting [this equation](#) into [this equation](#).

Note the sharp points in [\[link\]\(b\)](#), produced by the sharp peaks in the 1-D wavelets of [this previous figure](#) (Impulse and frequency responses of the 4-level tree of near-balanced 5,7-tap filters). These result in noticeable artefacts in reconstructed images when these wavelets are used. The smoother wavelets of [\[link\]\(c\)](#) are much better in this respect.

The 2-D frequency responses of the level 1 filters, derived from the LeGall 3,5-tap filters, are shown in figs [\[link\]](#) (in mesh form) and [\[link\]](#) (in contour form). These are obtained by substituting $z_1 = e^{i\omega_1}$ and $z_2 = e^{i\omega_2}$ into [\[link\]](#). [\[link\]](#) demonstrates that the 2-D frequency response is just the product of the responses of the relevant 1-D filters.

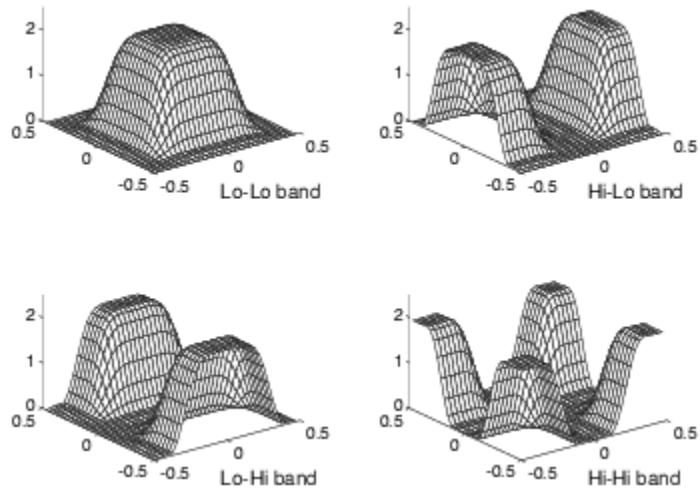


Mesh frequency response plots of the 2-D level 1 filters, derived from the LeGall 3,5-tap filters.



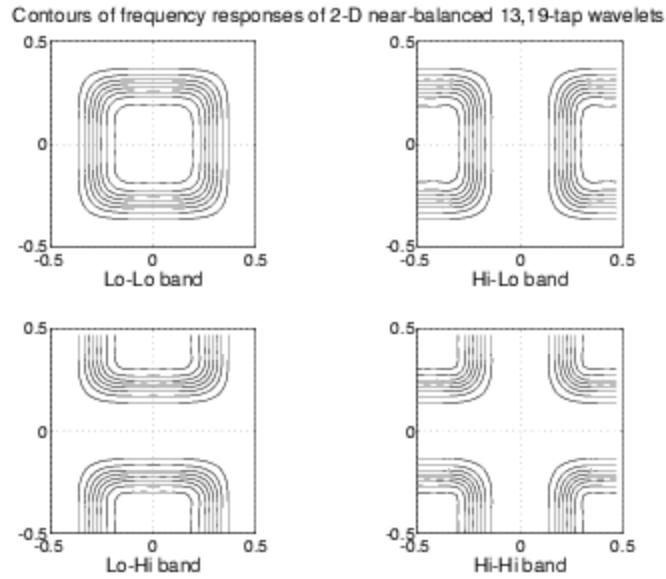
Contour plots of the frequency responses of [\[link\]](#).

[\[link\]](#) and [\[link\]](#) are the equivalent plots for the 2-D filters derived from the near-balanced 13,19-tap filters. We see the much sharper cut-offs and better defined pass and stop bands of these filters. The high-band filters no longer exhibit gain peaks, which are rather undesirable features of the LeGall 5-tap filters.



Mesh frequency response plots of the

2-D level 1 filters, derived from the near-balanced 13,19-tap filters.



Contour plots of the frequency responses of [\[link\]](#).

Compression Properties of Wavelets

We now look at how well the various wavelet filters perform in practice. We have used them in place of the [Haar transform](#) discussed earlier, and have measured the entropies and reconstructed the images from quantised coefficients.

In order to allow a fair comparison with the JPEG DCT results, we have modified the DWT quantising strategy to take advantage of the reduced visibility of the higher frequency wavelets. This approximately matches the effects achieved by the JPEG Q_{lum} matrix of [this previous equation](#). To achieve a high degree of compression we have used the following allocation of quantiser step sizes to the 4-level DWT bands:

Levels	Q_{step}
All bands at levels 3 and 4:	50
Hi-Lo and Lo-Hi bands at level 2:	50
Hi-Hi band at level 2:	100
Hi-Lo and Lo-Hi bands at level 1:	100
Hi-Hi band at level 1:	200

A similar compressed bit rate is produced by the 8×8 DCT when $Q_{\text{step}} = 5Q_{\text{lum}}$.

For reference, [\[link\]](#) compares the DCT and Haar transforms using these two quantisers. The rms errors between the reconstructed images and the original are virtually the same at 10.49 and 10.61 respectively, but the DCT

entropy of 0.2910 bit/pel is significantly lower than the Haar entropy of 0.3820 bit/pel. Both images display significant **blocking artefacts** at this compression level.



(a)



(b)

Reconstructions after coding using the 8×8 DCT (a) with $Q_{\text{step}} = 5Q_{\text{lum}}$, and (b) using the Haar transform with Q_{step} from [\[link\]](#).

[\[link\]](#) shows the reconstructed images for the following four DWTs using the quantiser of [\[link\]](#):

- The LeGall 3,5-tap filters: H_0, H_1 and G_0, G_1 from [these previous equations](#) and [these previous equations](#) from our discussion of good filters/wavelets.
- The inverse-LeGall 5,3-tap filters: [these previous equations](#) and [these previous equations](#) from our discussion of good filters/wavelets with H_0, H_1 and G_0, G_1 swapped.
- The near-balanced 5,7-tap filters: substituting $Z = \frac{1}{2}(z + z^{-1})$ into [this equation](#) from Good Filter/Wavelets.
- The near-balanced 13,19-tap filters: substituting [this previous equation](#) into [this equation](#).

We see that the LeGall 3,5-tap filters ([\[link\]\(a\)](#)) produce a poor image, whereas the other three images are all significantly better. The poor image

is caused by the roughness of the LeGall 5,3-tap filters (shown in this previous [figure](#)) which are used for reconstructing the image when the 3,5-tap filters are used for analysing the image. When these filters are swapped, so that the reconstruction filters are the 3,5-tap ones of [this figure](#), the quality is greatly improved ([\[link\]](#)(b)).

The near-balanced 5,7-tap filters ([\[link\]](#)(c)) produce a relatively good image but there are still a few bright or dark point-artefacts produced by the sharp peaks in the wavelets (shown in this previous [figure](#)). The smoother 13,19-tap wavelets ([see this figure](#)) eliminate these, but their longer impulse responses tend to cause the image to have a slightly **blotchy** or **mottled** appearance.



(a)



(b)



(c)

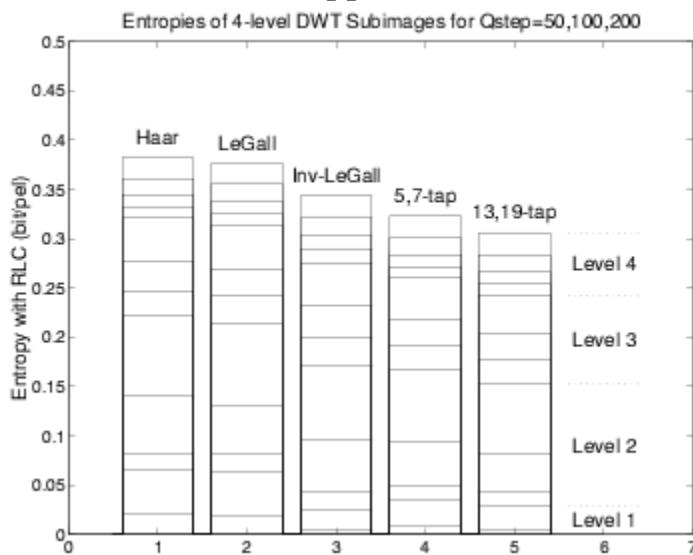


(d)

Reconstructions after coding using Q_{step} from [\[link\]](#) with (a) the LeGall 3,5-tap filters, (b) the inverse-

LeGall 5,3-tap filters, (c) the near-balanced 5,7-tap filters, and (d) the near-balanced 13,19-tap filters.

[link] shows the entropies (with RLC) of the separate subimages of the 4-level DWT for the Haar filter set and the four filter sets of [link]. Q_{step} is defined by [link] and it is particularly noticeable how the higher step sizes at levels 1 and 2 substantially reduce the entropy required to code these levels (compare with this [previous figure](#)). In fact the Hi-Hi band at level 1 is not coded at all! The reduction of entropy with increasing filter smoothness is also apparent.

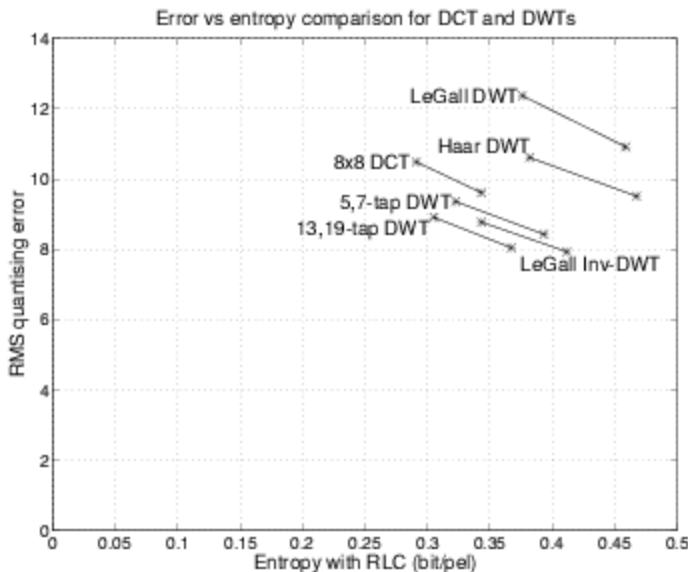


Entropies of 4-level DWT subimages using Q_{step} defined by [link], for five different wavelet filter pairs.

Note: We see that we have now been able to reduce the bit rate to around 0.3 bit/pel.

However measurement of entropy is not the whole story, as it is the tradeoff of entropy vs quantising error which is important. [\[link\]](#) attempts to show this trade-off by plotting rms quantising error (obtained by subtracting the reconstructed image from the original) versus the entropy for the 8×8 DCT and the five DWTs. To show the slope of the curves, the measurements are repeated with an 80% lower quantiser step-size, giving lower rms errors and higher entropies. The pair of points for each configuration are jointed by lines which indicate the slope of the rate-distortion curve.

Measurements at many more step sizes can be taken in order to give more compete rate-distortion curves if required.



RMS error vs. entropy for the 8×8 DCT and five wavelet filter pairs. For the DCT, $Q_{\text{step}} = 5Q_{\text{lum}}$ and $4Q_{\text{lum}}$; for the DWT, Q_{step} is 1.0 and 0.8 of the values in ([\[link\]](#)).

The good performance of the 13,19-tap filters is clear, but the inverse-LeGall filters do surprisingly well - showing that the poor smoothness of the analysis filters does not seem to matter. Correct ways to characterise

unbalanced filter sets to account properly for this phenomenon are still the subject of current research.

Note: What is clear is that when filters are unbalanced between analysis and reconstruction, the ones which give smoother wavelets **must** be used for reconstruction.

Finally, in these tests, the assessments of subjective image quality approximately match the assessments based on rms errors. However this is not always true and one must be careful to backup any conclusions from rms error measurements with at least some subjective tests.

Motion-Compensated Predictive Coding

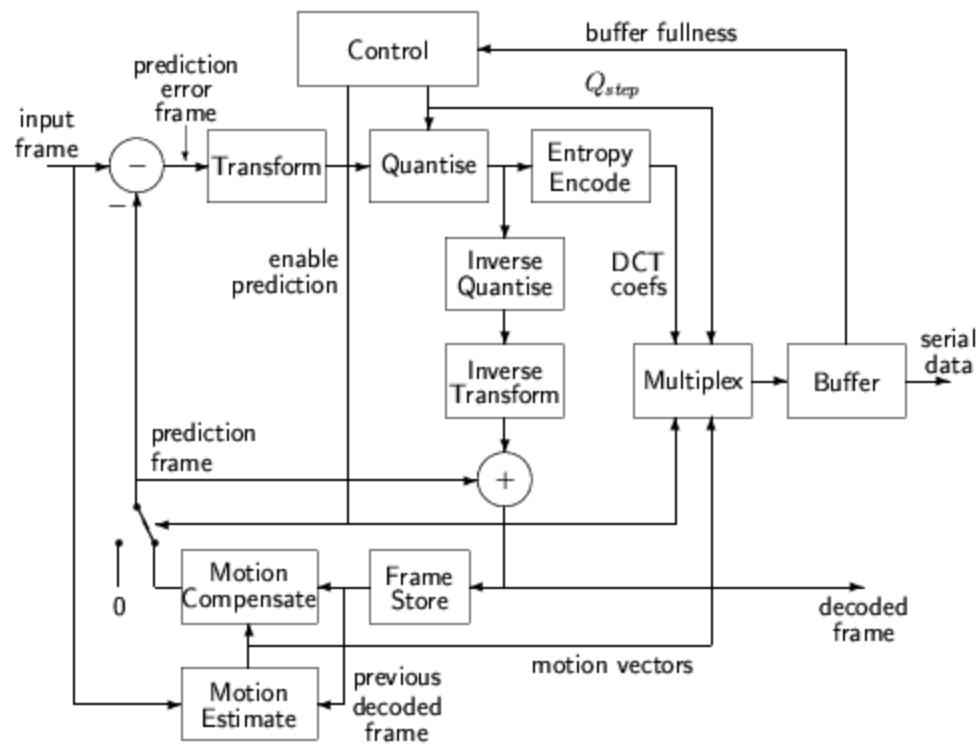
Video compression is concerned with coding image sequences at low bit rates. In an image sequence, there are typically high correlations between consecutive frames of the sequence, in addition to the spatial correlations which exist naturally within each frame.

Video coders aim to take maximum advantage of **interframe** temporal correlations (between frames) as well as **intraframe** spatial correlations (within frames).

Motion-Compensated Predictive Coding

Motion-compensated predictive coding (MCPC) is the technique that has been found to be most successful for exploiting interframe correlations.

[\[link\]](#) shows the basic block diagram of a MCPC video encoder.



Motion compensated prediction coding (MCPC) video encoder.

The transform, quantise, and entropy encode functions are basically the same as those employed for still image coding. The first frame in a sequence is coded in the normal way for a still image by switching the prediction frame to zero.

For subsequent frames, the input to the transform stage is the difference between the **input** frame and the **prediction** frame, based on the previous decoded frame. This difference frame is usually known as the **prediction error frame**.

The purpose of employing predictions is to reduce the energy of the prediction error frames so that they have lower entropy after transformation and can therefore be coded with a lower bit rate.

If there is motion in the sequence, the prediction error energy may be significantly reduced by **motion compensation**. This allows regions in the prediction frame to be generated from **shifted** regions from the previous decoded frame. Each shift is defined by a **motion vector** which is transmitted to the decoder in addition to the coded transform coefficients. The motion vectors are usually entropy coded to minimise the extra bit rate needed to do this.

The multiplexer combines the various types of coded information into a single serial bit stream, and the buffer smoothes out the fluctuations in bit rate caused by varying motion in the sequence and by scene changes. The controller adjusts coding parameters (such as the quantiser step size) in order to maintain the buffer at approximately half-full, and hence it keeps the mean bit rate of the encoder equal to that of the channel.

Decoded frames are produced in the encoder, which are identical to those generated in the decoder. The decoder comprises a buffer, de-multiplexer, and entropy decoder to invert the operations of the equivalent encoder blocks, and then the decoded frames are produced by the part of the encoder

loop comprising the inverse quantiser, inverse transform, adder, frame store, motion compensator and switch.

H.261 is a CCITT standard for video encoding for video-phone and video conferencing applications. Video is much more important in a multi-speaker conferencing environment than in simple one-to-one conversations. H.261 employs coders of the form shown in [\[link\]](#) to achieve reasonable quality head-and-shoulder images at rates down to 64 kb/s (one ISDN channel). A demonstration of H.261 coding at 64 and 32 kb/s will be shown.

A development of this, H.263, is now in current use, to allow the bit rate to be reduced down to about 20 kb/s, without too much loss of quality, for modems and mobile channels. This uses some of the more advanced motion methods from MPEG (see later), and it also forms the basis for baseline MPEG-4 coders.

Motion Estimation

Motion estimation is the processes which generates the motion vectors that determine how each motion compensated prediction frame is created from the previous frame.

Block Matching (BM) is the most common method of motion estimation. Typically each macroblock (16×16 pels) in the new frame is compared with shifted regions of the same size from the previous decoded frame, and the shift which results in the minimum error is selected as the best motion vector for that macroblock. The motion compensated prediction frame is then formed from all the shifted regions from the previous decoded frame.

BM can be very computationally demanding if all shifts of each macroblock are analysed. For example, to analyse shifts of up to $\pm(15)$ pels in the horizontal **and** vertical directions requires $31 \times 31 = 961$ shifts, each of which involves $16 \times 16 = 256$ pel difference computations for a given macroblock. This is known as **exhaustive search BM**.

Significant savings can be made with **hierarchical BM**, in which an approximate motion estimate is obtained from exhaustive search using a lowpass subsampled pair of images, and then the estimate is refined by a small local search using the full resolution images. Subsampling 2:1 in each direction reduces the number of macroblock pels **and** the number of shifts by 4:1, producing a computational saving of 16:1!

There are many other approaches to motion estimation, some using the frequency or wavelet domains, and designers have considered scope to invent new methods since this process does not need to be specified in coding standards. The standards need only specify how the motion vectors should be interpreted by the decoder (a much simpler process). Unfortunately, we do not have time to discuss these other approaches here.

The MPEG Standard

As a sequel to the JPEG standards committee, the Moving Picture Experts Group (MPEG) was set up in the mid 1980s to agree standards for video sequence compression.

Their first standard was MPEG-1, designed for CD-ROM applications at 1.5 Mb/s, and their more recent standard, MPEG-2, is aimed at broadcast quality TV signals at 4 to 10 Mb/s and is also suitable for high-definition TV (HDTV) at 20 Mb/s. We shall not go into the detailed differences between these standards, but simply describe some of their important features. MPEG-2 is used for digital TV and DVD in the UK and throughout the world.

MPEG coders all use the MCPC structure of [this previous figure](#), and employ the 8×8 DCT as the basic transform process. So in many respects they are similar to H.261 coders, except that they operate with higher resolution frames and higher bit rates.

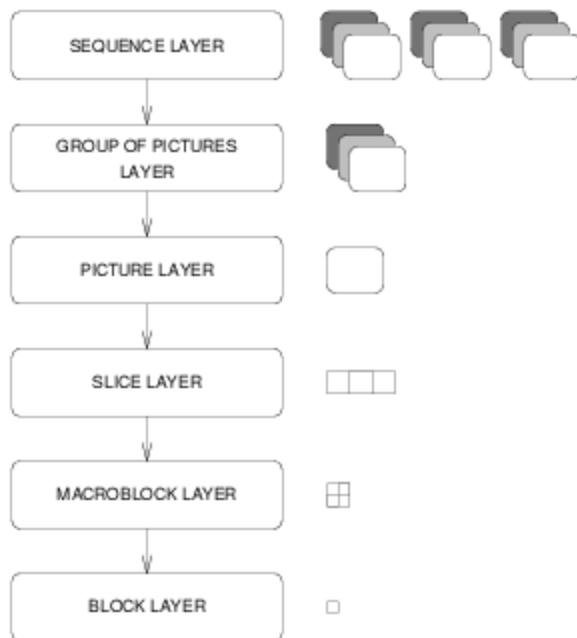
The main difference from H.261 is the concept of a Group of Pictures (GOP) Layer in the coding hierarchy, shown in [\[link\]](#). However we describe the other layers first:

- The Sequence Layer contains a complete image sequence, possibly hundreds or thousands of frames.
- The Picture Layer contains the code for a single frame, which may either be coded in absolute form or coded as the difference from a predicted frame.
- The Slice Layer contains one row of macroblocks (16×16 pels) from a frame. (48 macroblocks give a row 768 pels wide.)
- The Macroblock Layer contains a single macroblock -- usually 4 blocks of luminance, 2 blocks of chrominance and a motion vector.
- The Block Layer contains the DCT coefficients for a single 8×8 block of pels, coded almost as in JPEG using zig-zag scanning and run-amplitude Huffman codes.

The GOP Layer contains a small number of frames (typically 12) coded so that they can be decoded completely as a unit, without reference to frames

outside of the group. There are three types of frame:

- **Intra coded frames (I)** -- which are coded as single frames as in JPEG, without reference to any other frames.
- **Predictive coded frames (P)** -- which are coded as the difference from a motion compensated prediction frame, generated from an earlier I or P frame in the GOP.
- **Bi-directional coded frames (B)** -- which are coded as the difference from a bi-directionally interpolated frame, generated from earlier and later I or P frames in the sequence (with motion compensation).

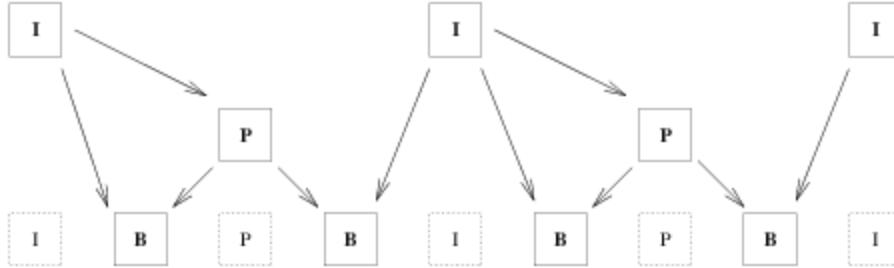


MPEG Layers.

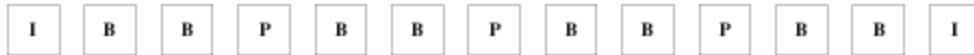
The main purpose of the GOP is to allow editing and splicing of video material from different sources and to allow rapid forward or reverse searching through sequences. A GOP usually represents about half a second of the image sequence.

[\[link\]](#) shows a typical GOP and how the coded frames depend on each other. The first frame of the GOP is always an I frame, which may be

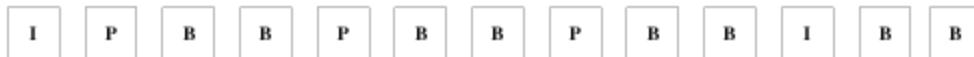
decoded without needing data from any other frame. At regular intervals through the GOP, there are P frames, which are coded relative to a prediction from the I frame or previous P frame in the GOP. Between each pair of I / P frames are one or more B frames.



A typical Group of Pictures (GOP).



Transmission order of a GOP.



GOP Layer -- Intra (I), Predicted (P) and Bi-directional (B) frames.

The I frame in each GOP requires the most bits per frame and provides the initial reference for all other frames in the GOP. Each P frame typically requires about one third of the bits of an I frame, and there may be 3 of these per GOP. Each B frame requires about half the bits of a P frame and there may be 8 of these per GOP. Hence the coded bits are split about evenly between the three frame types.

B frames require fewer bits than P frames mainly because bi-directional prediction allows uncovered background areas to be predicted from a subsequent frame. The motion-compensated prediction in a B frame may be forward, backward, or a combination of the two (selected in the macroblock layer). Since no other frames are predicted from them, B frames may be coarsely quantised in areas of high motion and comprise mainly motion prediction information elsewhere.

In order to keep all frames in the coded bit stream causal, B frames are always transmitted **after** the I/P frames to which they refer, as shown at the bottom of [\[link\]](#) .

One of the main ways that the H.263 (enhanced H.261) standard is able to code at very low bit rates is the incorporation of the B frame concept.

Considerable research work at present is being directed towards more sophisticated motion models, which are based more on the outlines of objects rather than on simple blocks. These will form the basis of extensions to the new low bit-rate video standard, MPEG-4 (MPEG-3 is an **audio** coding standard).